Hardware Acceleration of the Tate Pairing in Characteristic Three *

P. Grabher[1] and D. Page[2]

[1] Institute for Applied
Information Processing and Communications,
Graz University of Technology,
Inffeldgasse 16a,
A-8010 Graz,
Austria.
grabherp@sbox.tugraz.at
[2] Department of Computer Science,
University of Bristol,
Merchant Venturers Building,
Woodland Road,
Bristol, BS8 1RB,
United Kingdom.
page@cs.bris.ac.uk

Abstract. Although identity based cryptography offers many functional advantages over conventional public key alternatives, the computational costs are significantly greater. The core computational task is evaluation of a bilinear map, or pairing, over elliptic curves. In this paper we prototype and evaluate polynomial and normal basis field arithmetic on an FPGA device and use it to construct a hardware accelerator for pairings over fields of characteristic three. The performance of our prototype improves roughly ten-fold on previous known hardware implementations and orders of magnitude on the fastest known software implementation. As a result we reason that even on constrained devices one can usefully evaluate the pairing, a fact that gives credence to the idea that identity based cryptography is an ideal partner for identity aware smart-cards.

Keywords. Identity Based Encryption, Pairing, Elliptic Curve, FPGA.

1 Introduction

The notion of identity based cryptography was first proposed by Shamir [25] in 1984. Essentially it allows a user identity, an arbitrary string, to play the role of

* The work described in this paper has been supported in part by the European Commission through the IST Programme under Contract IST-2002-507932 ECRYPT. The information in this document reflects only the author's views, is provided as is and no guarantee or warranty is given that the information is fit for any particular purpose. The user thereof uses the information at its sole risk and liability

a public key rather than have the key derived from a relationship with private information as would be the case in traditional schemes such as RSA. This can vastly reduce the amount of certification infrastructure required and generally presents a rich set of functional and security characteristics that are difficult or impossible to realise with other solutions. The first efficient Identity Based Encryption (IBE) scheme was presented by Boneh and Franklin [8] who followed the idea of Sakai, Ohgishi and Kasahara [23] in basing their scheme on bilinear maps, or pairings, over elliptic curves.

Although pairing and identity based cryptography has sparked a wealth of research into cryptographic schemes [7,11] and proof techniques, it has remained an ongoing task to reduce the computational cost that underpins such work. Theorists have generally worked under the gross assumption that a pairing takes around ten times as long to compute than the major computational task in elliptic curve cryptography (ECC), the point multiplication. Although in reality this ratio is significantly lower, the cost of pairing evaluation still constitutes a major hurdle. This is particularly true in constrained environments such as smart-cards which, due to their use as identity-aware tokens, seem a natural partner for identity based cryptography.

Recently, Gemplus announced that it had developed a smart-card hosted IBE implementation in partnership with the market leaders Voltage Security [27]. Although details are scarce, it seems probable that they use an existing core for $\mathbb{F}_p$ arithmetic to accelerate a software implementation of the BKLS algorithm [4]. This seems the natural decision given the increasing flexibility in parameterisation [3,5,19] and expertise related to implementing arithmetic in $\mathbb{F}_p$ accumulated from building conventional ECC and RSA based systems. However, in the short term at least it is attractive to consider working over fields of characteristic three since when parameterised using suitable supersingular elliptic curves, the resulting system boasts a higher security multiplier [12], given by the MOV embedding degree [20]. Additionally, there are some specialised, high-performance algorithms for computing pairings in this context: the Duursma-Lee algorithm [10], recently improved upon by Kwon [18] and Barreto et al. [2], uses a closed formula for the pairing which is efficient as long as the underlying field arithmetic in $\mathbb{F}_{3^m}$ is also efficient. To this end, previous work has considered the possibility of using polynomial [6, 22] and normal bases [13] to implement said arithmetic. However, such work has focused mainly on arithmetic performance rather than placing the designs in context to actually compute IBE related functions, the exception being Kerins, Popovici and Marnane [17] who quote estimated timings for FPGA hosted pairing hardware using a BKLS style algorithm.

In this paper, our main aims are three-fold: to evaluate the performance and cost of constructing hardware polynomial and normal basis arithmetic in $\mathbb{F}_{3^m}$; to investigate the possibility of construct a hardware accelerator that is small enough for use in constrained environments; to prove pairings over $\mathbb{F}_{3^m}$ using the closed form family of algorithms are a viable alternative to the use of $\mathbb{F}_p$ and BKLS. We prototype our work on an FPGA device and present experimental results of the performance and cost comparisons with previous work in this area.

Algorithm 1: The Duursma-Lee algorithm [10] for calculating the Tate pairing in characteristic three.

Input : point $P = (x_1, y_1)$, point $Q = (x_2, y_2)$
Output : $f_P(\phi(Q)) \in \mathbb{F}^*_{q^6}/\mathbb{F}^*_{q^3}$

$f \leftarrow 1$
for $i = 1$ **upto** m **do**
  $x_1 \leftarrow x_1^3$
  $y_1 \leftarrow y_1^3$
  $\mu \leftarrow x_1 + x_2 + b$
  $\lambda \leftarrow -y_1 y_2 \sigma - \mu^2$
  $g \leftarrow \lambda - \mu\rho - \rho^2$
  $f \leftarrow f \cdot g$
  $x_2 \leftarrow x_2^{1/3}$
  $y_2 \leftarrow y_2^{1/3}$
return f

We organise our work as follows: in Section 2 we give an overview of pairings before using Section 3 to present details of arithmetic in $\mathbb{F}_{3^m}$. We then discuss the details of our accelerator architecture and present experimental results in Section 4 before concluding in Section 5.

2 An Introduction to Pairings

To provide a concrete case for discussion, we use the example of pairings where the base field is of characteristic three, i.e. $\mathbb{F}_q$ where $q = 3^m$. To allow investigation of both polynomial and normal bases we consider cases $m = 97$ and $m = 89$ respectively. Let E be an elliptic curve over a finite field $\mathbb{F}_q$, and let $\mathcal{O}$ denote the identity element of the associated group of rational points $E(\mathbb{F}_q)$. For a positive integer $l | \#E(\mathbb{F}_q)$ coprime to q, let $\mathbb{F}_{q^k}$ be the smallest extension field of $\mathbb{F}_q$ which contains the l-th roots of unity in $\overline{\mathbb{F}}_q$. Also, let $E(\mathbb{F}_q)[l]$ denote the subgroup of $E(\mathbb{F}_q)$ of all points of order dividing l, and similarly for the degree k extension of $\mathbb{F}_q$. Setting $k = 6$, we parameterise $\mathbb{F}_{q^6}$ as the quadratic extension $\mathbb{F}_{q^6} = \mathbb{F}_{q^3}[\sigma]/(\sigma^2 + 1)$. Further, we set $\mathbb{F}_{q^3} = \mathbb{F}_q[\rho]/(\rho^3 - \rho - 1)$. For efficient arithmetic in these fields, we to the work of Granger et al. [14].

Our choice of prime values for m is motivated by well known security considerations; both our choices offer an security level which is roughly equivalent to $800--900$-bit RSA. Using a polynomial basis with $m = 97$ provides us with a curve which is well known in the literature and hence a good reference against which to compare our results. However, one can only construct a type-two normal basis where $2m + 1$ is also prime: the most efficient type-one basis is never available. This limits our choices significantly. We settled on $m = 89$ since it is the closest choice to $m = 97$ for which affords a suitable parameterisation. For both our choices of m, we use the curve $E : Y^2 = X^3 - X + 1$. In the case

Algorithm 2: The Kwon-BGOS algorithm [18] for calculating the Tate pairing in characteristic three.

Input : point $P = (x_1, y_1)$, point $Q = (x_2, y_2)$
Output : $f_P(\phi(Q)) \in \mathbb{F}^*_{q^6}/\mathbb{F}^*_{q^3}$

$f \leftarrow 1$
$x_2 \leftarrow x_2^3$
$y_2 \leftarrow y_2^3$
$d \leftarrow mb$
for $i = 1$ **upto** m **do**
  $x_1 \leftarrow x_1^9$
  $y_1 \leftarrow y_1^9$
  $\mu \leftarrow x_1 + x_2 + d$
  $\lambda \leftarrow y_1 y_2 \sigma - \mu^2$
  $g \leftarrow \lambda - \mu\rho - \rho^2$
  $f \leftarrow f^3 \cdot g$
  $y_2 \leftarrow -y_2$
  $d \leftarrow d - b$
return f

of $m = 89$ this has an unattractively large cofactor [13]: this parameterisation problem alone might be viewed as a reason not to use a normal basis representation; we stress that our aim in selecting these parameters is performance and cost comparison only.

The Reduced Tate Pairing For a thorough treatment of the following, we refer the reader to [4] and also [12], and to [24] for an introduction to divisors. The reduced Tate pairing of order l is the map

$$e_l : E(\mathbb{F}_q)[l] \times E(\mathbb{F}_{q^k})[l] \to \mathbb{F}^*_{q^k}/(\mathbb{F}^*_{q^k})^l,$$

given by $e_l(P, Q) = f_{P,l}(\mathcal{D})$. Here $f_{P,l}$ is a function on E whose divisor is equivalent to $l(P) - l(\mathcal{O})$, $\mathcal{D}$ is a divisor equivalent to $(Q) - (\mathcal{O})$, whose support is disjoint from the support of $f_{P,l}$, and $f_{P,l}(\mathcal{D}) = \prod_i f_{P,l}(P_i)^{a_i}$, where $\mathcal{D} = \sum_i a_i P_i$. It satisfies the following properties:

- For each $P \neq \mathcal{O}$ there exists $Q \in E(\mathbb{F}_{q^k})[l]$ such that $e_l(P, Q) \neq 1 \in \mathbb{F}^*_{q^k}/(\mathbb{F}^*_{q^k})^l$ (*non-degeneracy*).
- For any integer n, $e_l([n]P, Q) = e_l(P, [n]Q) = e_l(P, Q)^n$ for all $P \in E(\mathbb{F}_q)[l]$ and $Q \in E(\mathbb{F}_{q^k})[l]$ (*bilinearity*).
- Let $L = hl$. Then $e_l(P, Q)^{(q^k-1)/l} = e_L(P, Q)^{(q^k-1)/L}$.
- It is efficiently computable.

The non-degeneracy condition requires that Q is not a multiple of P, i.e. that Q is in some order l subgroup of $E(\mathbb{F}_{q^k})$ disjoint from $E(\mathbb{F}_q)[l]$. When one computes

$f_{P,l}(\mathcal{D})$, the value obtained belongs to the quotient group $\mathbb{F}_{q^k}^*/(\mathbb{F}_{q^k}^*)^l$, and not $\mathbb{F}_{q^k}^*$. In this quotient, for a and b in $\mathbb{F}_{q^k}^*$, $a \sim b$ if and only if there exists $c \in \mathbb{F}_{q^k}^*$ such that $a = bc^l$. Clearly, this is equivalent to

$$a \sim b \text{ if and only if } a^{(q^k-1)/l} = b^{(q^k-1)/l},$$

and hence one ordinarily uses this value as the canonical representative of each coset. The isomorphism between $\mathbb{F}_{q^k}^*/(\mathbb{F}_{q^k}^*)^l$ and the elements of order l in $\mathbb{F}_{q^k}^*$ given by this exponentiation makes it possible to compute $f_{P,l}(Q)$ rather than $f_{P,l}(\mathcal{D})$.

The Modified Tate Pairing Duursma and Lee introduced their algorithm [10] in the context of pairings on a family of supersingular hyperelliptic curves. The performance of their method was improved upon by Kwon [18] and Barreto et al. [2] who also provide similar algorithms for other characteristics.

Let $q = 3^m$ and $E(\mathbb{F}_q) : Y^2 = X^3 - X + b$, with $b = \pm 1$, and let $P = (x_1, y_1)$ and $Q = (x_2, y_2)$ be points of order l. Let $\mathbb{F}_{q^3} = \mathbb{F}_q[\rho]/(\rho^3 - \rho - b)$, with $b = \pm 1$ depending on the curve equation, and let $\mathbb{F}_{q^6} = \mathbb{F}_{q^3}[\sigma]/(\sigma^2 + 1)$. Then the modified Tate pairing on E is the mapping $f_P(\phi(Q))$ where $\phi : E(\mathbb{F}_q) \to E(\mathbb{F}_{q^6})$ is the distortion map $\phi(x_2, y_2) = (\rho - x_2, \sigma y_2)$. The methods for computing the Duursma-Lee and Kwon-BGOS algorithms are shown in Algorithm 1 and Algorithm 2 respectively. Note that the final result is powered by $q^3 - 1$ to form a compatible result with the BKLS [4] algorithm.

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

The finite field $\mathbb{F}_{3^m}$ is isomorphic to $\mathbb{F}_3[X]/(p)$ and $\mathbb{F}_3(\alpha)$ where p is an irreducible polynomial of degree m in $\mathbb{F}_3[X]$ and α is a root of p. We will identify these three fields, but our notation will be tailored toward $\mathbb{F}_3(\alpha)$. In a polynomial basis $\mathbb{F}_3(\alpha)$ is regarded as an m-dimensional vector space over $\mathbb{F}_3$ with basis

$$(\alpha^0, \alpha^1, \ldots, \alpha^{m-1}) .$$

For an element $\hat{a} \in \mathbb{F}_3(\alpha)$ we will simply write the elements in a polynomial, or standard basis as

$$\hat{a} = \sum_{i=0}^{m-1} \hat{a}_i \cdot \alpha^i .$$

Arithmetic in a polynomial basis is fairly straightforward when based on conventional polynomial arithmetic. When discussing implementation of such arithmetic, it is often useful to denote elements as a vector of coefficients such as

$$\hat{a} = (\hat{a}_0, \hat{a}_1, \hat{a}_2, \ldots, \hat{a}_{m-1}) ,$$

so that physical operations such as shifting and rotation of coefficients is more naturally expressed. We use the notation $\hat{a}^{(i)}$ to denote the (left) rotation of the coefficients in such a vector by distance i. That is, we write

$$\hat{a}^{(i)} = (\hat{a}_{i+0}, \hat{a}_{i+1}, \hat{a}_{i+2}, \ldots, \hat{a}_{i+m-1}).$$

where in all cases, coefficient indices are reduced modulo m. Using this notation, $\hat{a}_j^{(i)}$ represents the j-th coefficient of the rotated element $\hat{a}^{(i)}$.

In a normal basis, things are slightly more involved. Given an irreducible polynomial p of degree m and with root α, the full set of roots of p in $\mathbb{F}_3(\alpha)$ is

$$\mathbb{B} = (\alpha, \alpha^3, \alpha^{3^2}, \ldots, \alpha^{3^{m-1}}).$$

If the elements of $\mathbb{B}$ are linearly independent then the set of roots forms a basis of $\mathbb{F}_3(\alpha)$ over $\mathbb{F}_3$ and this basis, p and α are all called normal. To construct such as basis, and the matrix M which determines how the multiplication operation works, we use the techniques of Granger et. al [13] based on work by Nöcker [21]. For an element $\bar{a} \in \mathbb{F}_3(\alpha)$ we write

$$\bar{a} = \sum_{i=0}^{m-1} \bar{a}_i \cdot \alpha^{3^i}$$

but again, for brevity, we often denote a normal basis field element using the coefficient vector and rotated coefficient vector notation as described above.

When using both polynomial and normal basis representations, we hold a polynomial over $\mathbb{F}_3$ of degree m as a $2m$ length vector of bits. Two sequential bits are used to hold each coefficient so that

$$a = (a_0^L, a_0^H, a_1^L, a_1^H, \ldots, a_{m-1}^L, a_{m-1}^H)$$

where

$$\begin{aligned} a_i^L &= a_i \mod 2 \\ a_i^H &= a_i \operatorname{div} 2 . \end{aligned}$$

For concreteness, we set the defining polynomial for our polynomial basis to $\alpha^{97} + \alpha^{16} + 2$ and the normal polynomial p that defines M in our normal basis to $\alpha^{89} + \alpha^{88} + 2\alpha^{87} + \alpha^{84} + 2\alpha^{83} + 2\alpha^{82} + \alpha^{81} + \alpha^{72} + \alpha^{71} + \alpha^{70} + 2\alpha^{69} + \alpha^{66} + 2\alpha^{65} + 2\alpha^{64} + \alpha^{63} + 2\alpha^{54} + \alpha^{35} + \alpha^{34} + 2\alpha^{33} + \alpha^{30} + 2\alpha^{29} + 2\alpha^{28} + \alpha^{27} + 2\alpha^{18} + 2\alpha^{17} + 2\alpha^{16} + \alpha^{15} + 2\alpha^{12} + \alpha^{11} + \alpha^{10} + 2\alpha^9 + 1$.

3.1 Addition and Subtraction

The most basic operations on field elements are addition and subtraction. These are made reasonably straightforward because they can be performed componentwise with no interaction with other coefficients. Given that our coefficients are held using two bits, we can construct cells for the required arithmetic using simple logical operations. Following Harrison et al. [15], the addition $r_i = a_i + b_i$ of two coefficients a_i and b_i can be specified using

$$\begin{aligned} r_i^H &= (a_i^L \vee b_i^L) \oplus t \\ r_i^L &= (a_i^H \vee b_i^H) \oplus t \end{aligned}$$

where

$$t = (a_i^L \vee b_i^H) \oplus (a_i^H \vee b_i^L).$$

Subtraction, and hence multiplication by two, are equally efficient since the negation of an element a simply swaps the bits a_i^H and a_i^L over and can therefore be implemented by the same function as addition.

3.2 Cubing and Cube Roots

When working in characteristic three, cubing is an important operation since curve and pairing arithmetic is often manipulated to utilise cubing rather than a more costly multiplication. In addition, the cube root operation is important in the Duursma-Lee algorithm if pre-computation is avoided.

When using a normal basis, the cube and cube root operations are very efficient in characteristic three: both can be achieved by cyclic shifting the coefficients in an elements so that for an element $\bar{a}$

$$\begin{aligned} \bar{a}^3 &= (\bar{a}_{m-1}, \bar{a}_0, \ldots, \bar{a}_{m-3}, \bar{a}_{m-2}), \\ \sqrt[3]{\bar{a}} &= (\bar{a}_1, \bar{a}_2, \ldots, \bar{a}_{m-1}, \bar{a}_0). \end{aligned}$$

Clearly these rotations can be easily implemented in a hardware circuit, where they reduce to wired permutation of bits with no actual computational overhead.

In a polynomial basis, cubing is a linear operation in the same way squaring is linear in characteristic two [6, 22]. That is, we have

$$(\hat{a}_i \alpha^i)^3 = \hat{a}_i^3 \alpha^{3i} = \hat{a}_i \alpha^{3i} \ .$$

Therefore, we can implement it using by simply thinning the coefficients, i.e. padding them with zeros, before performing a reduction. Cube root is somewhat more involved but since our chosen field is of the right form, we can utilise the method highlighted by Barreto [1]. Specifically, since our defining polynomial for $m = 97$ is $\alpha^{97} + \alpha^{16} + 2$ we have that $97 = 3u + 1$ and $16 = 3v + 1$ so that $u = 32$ and $v = 5$. Hence, for an element $\hat{a} = t_0 + t_1 + t_2$ where

$$\begin{aligned} t_0 &= \textstyle\sum_{i=0}^{u} \hat{a}_{3i} \alpha^i \\ t_1 &= \textstyle\sum_{i=0}^{u-1} \hat{a}_{3i+1} \alpha^i \\ t_2 &= \textstyle\sum_{i=0}^{u-1} \hat{a}_{3i+2} \alpha^i \end{aligned}$$

we have that

$$\sqrt[3]{\hat{a}} = t_0 + t_1^{\ll 2u+1} - t_1^{\ll u+v+1} + t_1^{\ll 2v+2} - 2t_2^{\ll u+1} - 2t_2^{\ll v+1}$$

given that for $t \in \mathbb{F}_{3^m}$, $t^{\ll n}$ denotes $t\alpha^n$, the value t shifted left by n coefficients and suitable reduced.

3.3 Multiplication

In addition to component-wise addition and subtraction, for normal basis multiplication we also require a component-wise multiplication of the form $r_i = a_i \cdot b_i$. This can be performed using similarly inexpensive logical operations

$$\begin{aligned} r_i^H &= (a_i^L \wedge b_i^H) \vee (a_i^H \wedge b_i^L) \\ r_i^L &= (a_i^L \wedge b_i^L) \vee (a_i^H \wedge b_i^H) \ . \end{aligned}$$

Armed with a function to perform this operation, we construct a general multiplication result of the form $\bar{c} = \bar{a} \cdot \bar{b}$ using

$$\bar{c}_k = \sum_{i=0}^{m-1} \bar{a}_{k+i} \cdot \sum_{j=0}^{m-1} M_{i,j} \cdot \bar{b}_{k+j}$$

where in all cases, coefficient indices are reduced modulo m. The sparse matrix M in this description is constructed from the normal polynomial p and essentially dictates how reduction behaves for the field. We developed a compiler that takes M and automatically produces circuitry to implement the three phases of the above formula: an addition phase to compute the terms $M_{i,j} \cdot \bar{b}_{k+j}$, keeping in mind that $M_{i,j} \in \{0, 1, 2\}$; a multiplication phase to multiply $\bar{a}_{k+i}$ by the summed terms; and accumulation phase sum all the multiplied terms and form $\bar{c}_k$. Such circuitry generates a single coefficient and hence requires m clock cycles to complete a multiply; we can place several of them working in parallel to accelerate the multiplication [13].

There has already been plenty of previous work dedicated to hardware polynomial basis multiplication methods in characteristic three [6, 17, 22]. We follow the approach of Bertoni et al. [6] in employing a digit-serial approach. In a similar way that a normal basis is scalable since we can utilise D parallel coefficient calculation circuits, a digit-serial multiplier allows us to scale the digit-size D in order to find a suitable balance between size and speed.

3.4 Inversion

Inversion is generally the most expensive operation when dealing with finite field arithmetic, so much so that in systems like ECC every effort is made to construct higher level operations so that inversion is not required. Due to the cost of constructing dedicated hardware for limited return, we implement inversion in software using our hardware for other operations in $\mathbb{F}_{3^m}$. To avoid the extra hardware cost described by Kerins et al. [17], we implement inversion using the relationship

$$a^{-1} = a^{3^m - 2}.$$

using a ternary expansion of the exponent since cubing operations are so inexpensive. In a polynomial basis this could be improved upon incrementally by using a translation of the standard binary Euclidean algorithm [15]. Since we only require inversion once in the final powering, we leave this issue for further work.

3.5 Exponentiation

Generally, we avoid exponentiation of pairing values by arbitrary exponents since one can use the bilinearity property to push the operation inside the pairing as a point multiplication which is more efficient, see the work of Granger et al. [14] for efficient methods in this area. However, we do need to consider the final powering

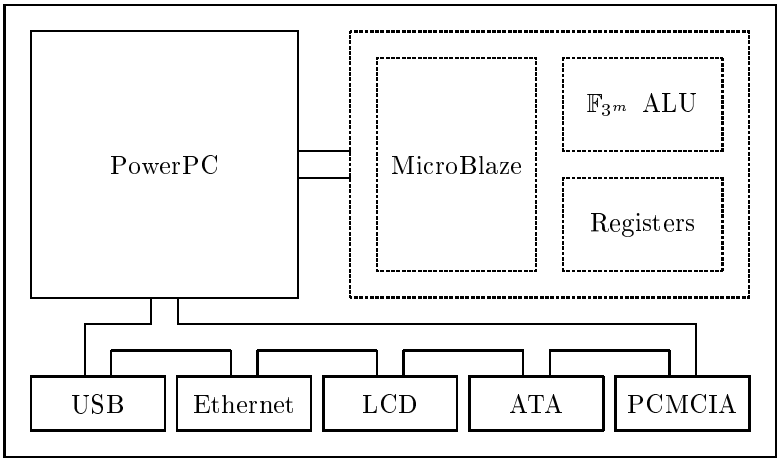

Fig. 1. A block diagram of our experimental architecture as hosted on a Xilinx ML300 prototyping device. Note that FPGA hosted elements are shown in dashed boxes while dedicated elements are shown in solid boxes.

of the pairing output by $q^3 - 1$ in order to yield a value compatible with BKLS. To power the pairing output f by the required exponent, we decompose the operation into

$$f^{3^{3m}} \cdot f^{-1}$$

the first term of which is simply three applications of the q-frobenius and the second is an inversion. Thanks to our field arithmetic, the inversion is reasonably efficient essentially because it can be done directly [14] rather than using an iterative method.

4 Architecture and Results

4.1 Architecture

Our design was realised using VHDL synthesised with a combination of Xilinx EDK 7.1 and ISE 7.1. Our experimental platform was a Xilinx ML300 prototyping board which hosts a Virtex-II PRO FPGA (XC2VP4FF672-6) device with 4928 slices. Our philosophy with this design was to treat the $\mathbb{F}_{3^m}$ arithmetic as a kind of co-processor, which is controlled by a more general purpose processor rather than hardwiring logic to directly compute the pairing. By swapping the co-processor we can provide arithmetic in either polynomial or normal bases; the FPGA size prevented making both available in one design. Since the instructions that are issued to the co-processor are executed synchronously, one might view this as a kind of instruction set extension. With this approach, we can easily implement other higher level operations based on the same field arithmetic, such as

$\mathbb{F}_{3^{97}}$ in Polynomial Basis					
	Slices	Cycles	Instructions	Speed	
				At 16 MHz	At 150 MHz
Add	112	3	1	-	-
Subtract	112	3	1	-	-
Multiply	946	28	1	-	-
Cube	128	3	1	-	-
Cube Root	115	3	1	-	-
Point Doubling	-	220	15	13.8μs	1.5μs
Point Tripling	-	52	9	3.3μs	0.4μs
Point Addition	-	366	22	22.9μs	2.4μs
Pairing					
Duursma-Lee	-	59946	7857	3746.6μs	399.4μs
Kwon	-	64602	9409	4037.6μs	430.7μs
Powering	-	4941	397	308.8μs	32.9μs
Total	4481	-	-	-	-
$\mathbb{F}_{3^{89}}$ in Normal Basis					
	Slices	Cycles	Instructions	Speed	
				At 16 MHz	At 85 MHz
Add	102	3	1	-	-
Subtract	102	3	1	-	-
Multiply	1505	48	1	-	-
Cube	0	3	1	-	-
Cube Root	0	3	1	-	-
Point Doubling	-	360	15	22.5μs	4.2μs
Point Tripling	-	72	9	4.5μs	0.8μs
Point Addition	-	606	22	37.9μs	7.1μs
Pairing					
Duursma-Lee	-	89046	7857	5563.3μs	1047.6μs
Kwon	-	93702	9409	5856.3μs	1102.4μs
Powering	-	7941	397	496.3μs	93.4μs
Total	4233	-	-	-	-

Table 1. Cost and performance characteristics of hardware based field, point and pairing arithmetic using polynomial and normal bases, clocked at low and maximum frequencies.

the ECC point multiplication over $E(\mathbb{F}_{3^m})$ which is also required in most pairing based schemes.

As such, we combine our arithmetic in $\mathbb{F}_{3^m}$ with a register file, backed by BlockRAM, of 32 registers each able to store an element of $\mathbb{F}_{3^m}$ which total under 1 kilobyte for our choices of m. We control this combined data-path with a Xilinx MicroBlaze soft-core, a 32-bit, 3-stage pipelined RISC processor which interfaces to the logic using the Fast Simplex Link (FSL) interface. The MicroBlaze code to control the co-processor was compiled using a re-targeted GCC tool-chain; we were able to achieve fast development times as a result. In short, the FPGA of our prototyping board is filled, as described by Figure 1, with what could be considered an embedded processor with a co-processor for arithmetic in $\mathbb{F}_{3^m}$. The obvious real-world analogy of this type of architecture is a smart-card with an associated co-processor.

4.2 Results

Having selected our fields for polynomial and normal bases so that they were as close as possible in size, we took the approach of utilising as equal an amount of the FPGA as possible to make comparison easier. Since our multiplier architecture in both cases allows for scalability by altering the digit-size D, we parameterised the polynomial basis multiplier with $D = 4$ and the normal basis multiplier with $D = 2$, choices that resulted in roughly the same area cost.

Table 1 shows the performance of our arithmetic and higher level functions at a modest clock speed that could be useful in a constrained environment and the fastest possible speed resulting from our synthesis results. A given arithmetic operation essentially requires $n + 2$ cycles, 1 cycle for the instruction fetch and decode, n for the execution and 1 to write-back the result into the register file. As well as cycle and wall-clock timings, we quote the number of instructions issued by the MicroBlaze core to the ALU. The area costs are inclusive of all system elements bar the instruction memory and register file which are backed by BlockRAM. The MicroBlaze core, FSL interface and debugging unit consumes roughly 1300 slices; the finite state machine to control the ALU consumes roughly 500 slices; the ALU logic consumes roughly 1700 slices depending on which elements are included. Note that our upper clock speed was bounded by 150 MHz since this was the maximum permitted by use of the MicroBlaze.

In terms of field arithmetic, we find that the polynomial basis representation is generally faster since although the cube and cube root circuits are more complex, the dominant feature was the multiplier. The critical path of the normal basis multiplier was far longer, forcing a lower clock speed, and the design much larger, meaning the polynomial multiplier could employ a larger, more efficient digit-size. Using these results and by simply looking at the algorithms, it is clear that the Duursma-Lee algorithm will be faster than that of Kwon-BGOS since although the later removes the need for a cube root in $\mathbb{F}_q$, it requires a cubing in $\mathbb{F}_{q^k}$. Thanks to the single-cycle cube root implementations, the cube in $\mathbb{F}_{q^k}$ will inevitably be slower. Table 1 confirms this by quoting results for evaluating

the pairing and for the final powering: one should view a pairing as being the combination of these two if the goal is compatibility with other algorithms.

Note that although the Kwon-BGOS algorithm is marginally slower it offers an attractive trade-off since we can omit the cube root logic from our design and save the associated slices. Also note that because of the fast cube root method of Barreto [1], the perceived advantage of a normal basis in being able to perform fast cube root operations is eliminated: the multiplier is the dominant cost as a result.

4.3 Analysis

In characteristic three, given our constrained setting, an efficient way to perform point multiplication using minimal pre-computation is to use the generalised non-adjacent form (GNAF) [9, 26], to construct a signed ternary expansion of the exponent $d \pmod{l}$. Such a representation is easy to compute and reduces the average density of non-zero trits from two thirds to one half. Using $\mathcal{A}$ to denote point addition and $\mathcal{T}$ to denote point tripling, the cost of an average point multiplication is

$$\frac{\log(d)}{\log(3)}\mathcal{T} + \frac{\log(d)}{2\log(3)}\mathcal{A}.$$

The Boneh-Franklin IBE scheme [8] is perhaps the most definitive example of the use of pairings within a concrete scheme. The trust authority or TA has a public key $P_{TA} = s \cdot P$ for a master secret s. A users public key is calculated from the string ID using a hash function as $P_{ID} = H_1(ID)$. The corresponding secret key is calculated by the TA as $S_{ID} = s \cdot P_{ID}$. To encrypt the message M, one selects a random r and computes the tuple

$$C = (U, V) = (r \cdot P, M \oplus H(e(P_{ID}, P_{TA})^r)),$$

to decrypt $C = (U, V)$, one computes the result

$$M = V \oplus H(e(S_{ID}, U)).$$

Considering our faster implementation using polynomial basis and Duursma-Lee algorithm with a modest clock speed of 16 MHz, we use $\mathcal{P}$ to denote the combination of pairing and final powering, $\mathcal{M}$ a point multiplication and $\mathcal{E}$ a field exponentiation. Using this notation we see that encryption costs $2\mathcal{M} + \mathcal{P}$ while decryption costs $\mathcal{P}$. Although we do not consider it as an option, given some extra storage the pairing required for encryption can be pre-computed which results in the cost being $\mathcal{M} + \mathcal{E}$. Using these costs and our timings from Table 1, we find that using our architecture we can perform Boneh-Franklin encryption in $\approx 7ms$ and decryption in $\approx 4ms$.

This performance is easily enough for practical applications since a given scheme will typically try to minimise the number of pairings executed. Thus, one can consider making a trade-off between performance and cost to reduce the device size. For example, we can remove the cube root logic as described

above and utilise the Kwon-BGOS algorithm. Additional optimisations in this direction include: reduction of the digit-size in our multiplication units; sharing a group of addition cells between the addition and multiplication operations, at the moment we place individual copies for each; improving the register allocation strategy or spilling values to the main memory so as to reduce the size of our register file containing $\mathbb{F}_q$ elements; and further turning of the MicroBlaze to eliminate the debug and RS232 logic used for development purposes only.

5 Conclusions

We have presented an accelerator for arithmetic in $\mathbb{F}_{3^m}$ and used it to implement the Tate pairing, a primitive which is of increasing importance in cryptographic schemes. Unlike previous work, we investigate both polynomial and normal basis representations of field elements and both the Duursma-Lee and Kwon-BGOS algorithms to compute the pairing. Our results demonstrate roughly a ten-fold improvement on the only other known hardware implementation [17] and orders of magnitude better than the fastest known software implementations.

The issue of size of slightly harder to quantify due to the use of FPGA as a target. Although our design is clearly still unrealistically large to place on a smart-card for example, we have demonstrated that our performance margin is so great, trade-offs that significantly reduce the area are viable. We leave the realisation of such optimisations for further work which might also include other marginal issues: acceleration of inversion in $\mathbb{F}_{3^m}$ using Euclidean techniques rather than by powering, perhaps by using extra hardware [17]; some comparison with existing, proprietary smart-card hosted implementations of the Tate pairing [27].

Acknowledgements

The authors would like to thank Rob Granger, Johann Großschädl, Elisabeth Oswald, Nigel Smart, Martijn Stam and Fré Vercauteren for invaluable help and support throughout the course of this work.

References

1. P.S.L.M. Barreto. A Note On Efficient Computation Of Cube Roots In Characteristic 3. In *Cryptology ePrint Archive*, Report 2004/305, 2004.
2. P.S.L.M. Barreto, S. Galbraith, C. O'hEigeartaigh and M. Scott. Efficient Pairing Computation on Supersingular Abelian Varieties. In *Cryptology ePrint Archive*, Report 2004/375, 2004.
3. P.S.L.M. Barreto, B. Lynn. M. Scott. Constructing Elliptic Curves with Prescribed Embedding Degree. In *Security in Communication Networks (SCN)*, Springer-Verlag LNCS 2576, 257–267, 2002.
4. P.S.L.M. Barreto, H. Kim, B. Lynn and M. Scott. Efficient Algorithms for Pairing-Based Cryptosystems. In *Advances in Cryptology (CRYPTO)*, Springer-Verlag LNCS 2442, 354–368, 2002.

5. P.S.L.M. Barreto and M. Naehrig. Pairing-Friendly Elliptic Curves of Prime Order. In *Cryptology ePrint Archive*, Report 2005/133, 2005.
6. 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)*, Springer-Verlag LNCS 2612, 158–175, 2003.
7. I.F. Blake G. Seroussi and N.P. Smart. *Advances in Elliptic Curve Cryptography.* Cambridge University Press, 2004.
8. D. Boneh and M. Franklin. Identity-Based Encryption from the Weil Pairing. In *SIAM Journal on Computing*, **32**(3), 586–615, 2003.
9. W. Clark and J. Liang. On Arithmetic Weight for a General Radix Representation of Integers. In *IEEE Transactions on Information Theory*, **19**, 823–826, 1973.
10. I. Duursma and H. Lee. Tate Pairing Implementation for Hyperelliptic Curves $y^2 = x^p - x + d$. In *Advances in Cryptology (ASIACRYPT)*, Springer-Verlag LNCS 2894, 111–123, 2003.
11. R. Dutta, R. Barua and P. Sarkar, Pairing-Based Cryptographic Protocols : A Survey. In *Cryptology ePrint Archive*, Report 2004/064, 2004.
12. S. Galbraith. Supersingular Curves in Cryptography. In *Advances in Cryptology (ASIACRYPT)*, Springer-Verlag LNCS 2248, 495–513, 2001.
13. R. Granger, D. Page and M. Stam. Hardware and Software Normal Basis Arithmetic for Pairing Based Cryptography in Characteristic Three. In *Cryptology ePrint Archive*, Report 2004/157, 2004.
14. R. Granger, D. Page and M. Stam. On Small Characteristic Algebraic Tori in Pairing-Based Cryptography. In *Cryptology ePrint Archive*, Report 2004/132, 2004.
15. K. Harrison, D. Page and N.P. Smart. Software Implementation of Finite Fields of Characteristic Three, for use in Pairing Based Cryptosystems. In *LMS Journal of Computation and Mathematics*, **5** (1), 181–193, London Mathematical Society, 2002.
16. T. Itoh and S. Tsujii. A Fast Algorithm for Computing Multiplicative Inverses in $GF(2^n)$ Using Normal Bases. In *Information and Computation* **78**, 171–177, 1988.
17. T. Kerins, E. Popovici and W.P. Marnane. Algorithms and Architectures for Use in FPGA Implementations of Identity Based Encryption Schemes. In *Field Programmable Logic and Application (FPL)*, Springer-Verlag LNCS 3203, 74–83, 2004.
18. S. Kwon. Efficient Tate Pairing Computation for Supersingular Elliptic Curves over Binary Fields. In *Cryptology ePrint Archive*, Report 2004/303, 2004.
19. A. Miyaji, M. Nakabayashi and S. Takano. New explicit conditions of elliptic curve traces for FR-reduction. In *IEICE Transactions on Fundamentals*, **E84-A** (5), 1234–1243, 2001.
20. A. Menezes, T. Okamoto, and S. A. Vanstone. Reducing Elliptic Curve Logarithms to Logarithms in a Finite Field. In *IEEE Transactions on Information Theory*, **39**, 1639–1646, 1993.
21. M. Nöcker. Data Structures for Parallel Exponentiation in Finite Fields. PhD Thesis, Universität Paderborn, 2001.
22. D. Page and N.P. Smart. Hardware Implementation of Finite Fields of Characteristic Three. In *4th Workshop on Cryptographic Hardware and Embedded Systems (CHES)*, Springer-Verlag LNCS 2523, 529–539, 2002.
23. R. Sakai, K. Ohgishi and M. Kasahara. Cryptosystems Based on Pairings. In *Symposium on Cryptography and Information Security (SCIS)*, 2000.
24. J. Silverman. *The Arithmetic of Elliptic Curves.* Springer-Verlag GTM 106, 1986.

25. A. Shamir. Identity-Based Cryptosystems and Signature Schemes. In *Advances in Cryptology (CRYPTO)*, Springer-Verlag LNCS 196, 47–53, 1985.
26. T. Takagi, S-M. Yen and B-C. Wu. Radix-r Non-Adjacent Form. In *Information Security Conference (ISC)*, Springer-Verlag LNCS 3225, 99–110, 2004.
27. Voltage Security, Press Release, *Gemplus Develops the World's First Identity-Based Encryption for Smart Cards.* Available from `http://www.voltage.com/about/pressreleases/PR041102.htm`.