Hardware and Software Normal Basis Arithmetic for Pairing Based Cryptography in Characteristic Three*

R. Granger, D. Page and M. Stam

Department of Computer Science,
University of Bristol,
Merchant Venturers Building,
Woodland Road,
Bristol, BS8 1UB,
United Kingdom.
{granger, page, stam}@cs.bris.ac.uk

Abstract. Although identity based cryptography offers a number of functional advantages over conventional public key methods, the computational costs are significantly greater. The dominant part of this cost is the Tate pairing which, in characteristic three, is best computed using the algorithm of Duursma and Lee. However, in hardware and constrained environments this algorithm is unattractive since it requires online computation of cube roots or enough storage space to pre-compute required results. We examine the use of normal basis arithmetic in characteristic three in an attempt to get the best of both worlds: an efficient method for computing the Tate pairing that requires no pre-computation and that may also be implemented in hardware to accelerate devices such as smart-cards. Since normal basis arithmetic in characteristic three has not received much attention before, we also discuss the construction of suitable bases and associated curve parameterisations.

1 Introduction

Since it was first suggested in 1984 by Shamir [29], the concept of identity based cryptography has been an attractive target for researchers because of the potential for simplifying conventional approaches to public key based systems. The central idea is that the public key for a user is simply their identity and is hence implicitly known to all other users. Identity is a flexible concept but an often used concrete example is that of an email address. Within this context, an identity for Alice might be the string

`alice@gmail.com`

* This work has been submitted to the IEEE for possible publication. Copyright may be transferred without notice, after which this version may no longer be accessible.

If Bob wants to send Alice secure email, he implicitly knows her email address and hence her identity and public key. Therefore he can encrypt the email to her without the same level of involvement from, for example, certificate and trust authorities. Recent advances have produced many workable instances of identity based cryptography which were driven mainly by the seminal work of Sakai, Ohgishi and Kasahara [27] and then by Boneh and Franklin [4] who both proposed concrete cryptographic schemes based on pairings on elliptic curves.

For such schemes to be feasible, one needs an efficiently computable pairing, or bilinear map. Generally the Tate pairing is selected for this task and is computed using one of several algorithms, such as that of Barreto, Kim, Lynn and Scott [2] or Duursma and Lee [6]. In characteristic three, which is often selected due to the bandwidth advantages of the parameterisation, the Duursma-Lee algorithm is certainly faster and recent work shows that it is probably the current best choice [10,28]. However, unlike the BKLS algorithm, it requires the computation of cube root operations which are relatively slow when using a polynomial basis representation. Although these cube roots can be pre-computed online using cubings, the space requirement is so significant, in constrained or hardware environments the method is unattractive as a result. As such, this feature means identity based cryptography is currently too expensive to run on naturally identity aware devices.

Normal bases [9] offer a convenient solution to this problem since cube and cube root operations are simply cyclic shifts of the coefficients of an element. Indeed, this seems to be the method Duursma and Lee envisaged using in their original paper, yet without investigating how [6]. Therein lies the catch: normal bases have a drawback in that multiplication is a more complex, and to a certain extent less studied operation compared to polynomial bases. To construct a high performance implementation of Duursma-Lee that takes advantage of the properties of normal bases, efficient multiplication methods are required. In characteristic three, we are not aware of any work that addresses this problem and we attempt to fill the gap in the literature with this paper. We present methods for constructing an appropriate basis, as well as software and hardware algorithms for arithmetic in it. Our performance and cost results indicate that using normal bases in software is unattractive due to the cost of multiplication, but that this cost can be avoided by parallel architectures in hardware. The use of hardware acceleration for normal bases thus makes identity based cryptography on constrained devices such as smart-cards a far more feasible proposition. Note that it is important to look at the cost of software *and* hardware methods: in a standard implementation it is unattractive to convert between representations where, for example, a smart-card might use hardware acceleration and a desktop computer, which might utilise software components only.

The paper is organised as follows. Firstly, we use Section 2 to overview the mathematics that underpin pairing based cryptography, paying particular attention to the Duursma-Lee algorithm that motivates the use of normal basis arithmetic. We then discuss how to construct such normal bases for characteristic three fields in Section 3 and suitable curve parameterisations in Section 4. Arithmetic with and representation of fields elements is presented in Section 5 before software and hardware implementation results are introduced in Sections 6 and 7 respectively. Finally we present some concluding remarks in Section 8.

2 Pairing Based Cryptography

The existence of efficiently computable, non-degenerate bilinear maps, or pairings, has allowed cryptographers to explore avenues of research which had previously been uninstantiable [29]. Originally used as a method of attack on elliptic curve cryptosystems [7, 21], the constructive potential of pairings based on elliptic curves is now well-studied, with many protocols and applications [4,5,13,16].

To support these applications much research activity has focused on developing efficient and easily implementable algorithms for their deployment [2,6,8]. Currently the most efficient method for pairing computation is the Duursma-Lee algorithm [6, 10], which applies to supersingular elliptic curves in characteristic three with MOV embedding degree six [21]. As well as pairing evaluation being approximately two and a half times faster than with the BKLS algorithm [2], the embedding degree of six, while not optimal in terms of the relative security requirements for discrete logarithm algorithms in characteristic three finite fields and elliptic curves, still offers a good security/efficiency trade-off for contemporary key-size recommendations.

In this section we give a brief summary of the mathematics underlying the reduced Tate pairing, as developed in [2,8], and give details of its Duursma-Lee variant, which we refer to as the modified Tate pairing.

2.1 The reduced Tate pairing

We first introduce some notation. Let E be an elliptic curve over a finite field $\mathbb{F}_q$, and let $\mathcal{O}_E$ denote the identity element of the associated group of rational points on $E(\mathbb{F}_q)$. For a positive integer l 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 $\mathbb{F}_q$. From an efficiency perspective, k is usually chosen to be even [2]. For a thorough treatment of the following, we refer the reader to [2] and also [8], and to [30] for an introduction to divisors. The reduced Tate pairing of order l is the map

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

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 $l(P) - l(\mathcal{O}_E)$, $\mathcal{D}$ is a divisor equivalent to $(Q) - (\mathcal{O}_E)$, 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 $P \neq \mathcal{O}_E$ 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)*.

Field	Field Polynomial	Curve	Order	MOV security
$\mathbb{F}_{3^{79}}$	$t^{79}+t^{26}+2$	$Y^2 = X^3 - X - 1$	$3^{79}+3^{40}+1$	750
$\mathbb{F}_{3^{97}}$	$t^{97}+t^{12}+2$	$Y^2 = X^3 - X + 1$	$(3^{97}+3^{49}+1)/7$	906
$\mathbb{F}_{3^{163}}$	$t^{163}+t^{80}+2$	$Y^2 = X^3 - X - 1$	$3^{163}+3^{82}+1$	1548
$\mathbb{F}_{3^{193}}$	$t^{193}+t^{12}+2$	$Y^2 = X^3 - X - 1$	$3^{193}-3^{97}+1$	1830
$\mathbb{F}_{3^{239}}$	$t^{239}+t^{24}+2$	$Y^2 = X^3 - X - 1$	$3^{239}-3^{120}+1$	2268
$\mathbb{F}_{3^{353}}$	$t^{353}+t^{142}+2$	$Y^2 = X^3 - X - 1$	$3^{353}+3^{177}+1$	3354

Table 1. A table of field definitions and curve equations.

- 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 $f_{P,l}(\mathcal{D})$ [2]. It also removes the need to compute the costly denominators in Miller's algorithm. We note that there are other more efficient methods to obtain a unique representative of the output coset [10]. However in this article we are concerned primarily with the computation of the modified Tate pairing.

2.2 The modified Tate pairing

Duursma and Lee introduced their algorithm in the context of pairings on a family of hyperelliptic curves. Restricting to the elliptic curve case, it applies to a family of supersingular curves in characteristic three, including those in Table 1.

The first column gives the field over which each curve is defined, and the second lists the corresponding irreducible polynomials defining the field extensions. The third lists the curve equations and the fourth gives the order of the subgroup used. The final column gives the bit-length of the smallest finite field into which the pairing value embeds, which is always a degree six extension in these cases. These parameter values were generated simply by testing which prime extension degrees suitable for efficient normal bases yielded orders for supersingular curves that are prime, or almost prime, i.e. those possessing a small cofactor.

Algorithm 2: The Duursma-Lee algorithms 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 $i = 1$ **to** 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

The modified Tate pairing improves on the reduced variant in three ways. Firstly, using the third property listed above, instead of computing the Tate pairing of order l, one uses the pairing of order $q^3 + 1$, which eliminates the need for any point additions in Miller's algorithm. Secondly, while this apparently increases the trit-length of the exponent by a factor of three, Duursma and Lee show that the divisor computed when processing three trits at a time has a very simple form, and hence no losses are incurred. Lastly, they provide a closed form expression for the pairing, thus simplifying implementations.

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 method for computing this is shown in Figure 2. Note that in each loop, one must compute two cubings and two cube roots, in contrast to the BKLS algorithm where one must compute four cubings.

Alternatively, if memory size is not an issue, one can pre-compute the cube roots in reverse order as successive cubings [10]. With this strategy, Algorithm 2 is considerably more efficient than BKLS in characteristic three, and indeed than the BKLS algorithm for even and large characteristic curves of comparable order. To maintain this efficiency when the pre-computation of cube roots is not viable, such as in constrained or hardware environments, it is vital that one can perform *both* cubings and cube roots efficiently. This is precisely why normal bases are well suited to pairing based applications, since both a cubing and a cube root are simply cyclic shifts of the vector of $\mathbb{F}_3$ elements representing an element in the extension field.

3 Notation and Construction of Bases

The finite field $\mathbb{F}_{3^m}$ is isomorphic to $\mathbb{F}_3[X]/(f)$ and $\mathbb{F}_3(\alpha)$ where f is an irreducible polynomial of degree m in $\mathbb{F}_3[X]$ and α is a root of f. We will identify these three fields, but our notation will be tailored towards $\mathbb{F}_3(\alpha)$. 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 $a \in \mathbb{F}_3(\alpha)$ we will simply write the elements in a polynomial, or standard basis as

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

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 f of degree m and with root α, the full set of roots of f 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 form a basis of $\mathbb{F}_3(\alpha)$ over $\mathbb{F}_3$ and this basis, f and α are all called normal. For an element $a \in \mathbb{F}_3(\alpha)$ we write

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

but again, for brevity, we often denote a normal basis field element a using the coefficient vector $\bar{a}$ and rotated coefficient vectors as described above.

The main advantage of a normal basis is that it allows fast application of the Frobenius map and its inverse, i.e. cubing and taking cube roots. Indeed, let $a \in \mathbb{F}_{3^m}$ be represented by $\bar{a}$, then for any $i \in \mathbb{Z}$ the element a^{3^i} is represented by $\bar{a}^{(i)}$, i.e., applying Frobenius only takes a rotation.

Normal basis multiplication is more complicated, but a common technique is based on a so-called multiplication matrix. Let $a, b \in \mathbb{F}_{3^m}$ and let $c = a \cdot b$. Let

$\alpha^{1+3^l} = \sum_{k=0}^{m-1} d_{lk}\alpha^{3^k}$, then

$$\begin{aligned}
\sum_{k=0}^{m-1} \bar{c}_k \alpha^{3^k} &= (\sum_{i=0}^{m-1} \bar{a}_i \alpha^{3^i})(\sum_{j=0}^{m-1} \bar{b}_j \alpha^{3^j}) \\
&= \sum_{i=0}^{m-1} \sum_{j=0}^{m-1} \bar{a}_i \bar{b}_j (\alpha^{1+3^{i-j}})^{3^j} \\
&= \sum_{i=0}^{m-1} \sum_{j=0}^{m-1} \bar{a}_i \bar{b}_j (\sum_{k=0}^{m-1} d_{i-j,k} \alpha^{3^k})^{3^j} \\
&= \sum_{i=0}^{m-1} \sum_{j=0}^{m-1} \sum_{k=0}^{m-1} \bar{a}_i \bar{b}_j d_{i-j,k} \alpha^{3^{k+j}} \\
&= \sum_{k=0}^{m-1} (\sum_{i=0}^{m-1} \sum_{j=0}^{m-1} \bar{a}_{i+k} \bar{b}_{j+k} d_{i-j,-j}) \alpha^{3^k} \\
&= \sum_{k=0}^{m-1} (\bar{a}^{(k)} M (\bar{b}^{(k)})^T) \alpha^{3^k} \quad ,
\end{aligned}$$

where for the penultimate equality the transformation $i = i' + k', j = j' + k', k = -j'$ has been used and the matrix M is defined by $M_{ij} = d_{j-i,-i}$. Note that the transformation in the penultimate step has been chosen such that the matrix is the same for all coefficients of $\bar{c}$; all that is required are rotations of $\bar{a}$ and $\bar{b}$.

The computation of the matrix M given a normal polynomial f is relatively straightforward. Below, we follow the description of IEEE P1353 [14, Annex A]. The matrix M follows easily once the matrix (d_{ij}) is known. The latter matrix gives the linear transformation that takes $\bar{a}$ and outputs the normal representation for αa for an arbitrary element a. In a polynomial basis multiplication by α boils down to multiplication of $\hat{a}$ by the companion matrix of f. To determine the matrix (d_{ij}), the standard solution is to first linearly transform from a normal basis representation $\bar{a}$ to a polynomial basis representation $\hat{a}$, multiply with the companion matrix and transform back (again linearly) to the normal basis.

The number of non-zero entries in the multiplication matrix M associated with the basis is in some sense a measure of complexity: this number is often denoted as C_N in the literature. For a random normal polynomial f the matrix M will be fairly dense. Using specially constructed normal bases it is possible to ensure that the matrix will be sparse.

The best known normal basis are based on Gauß periods. References to the development of the theory of normal bases based on Gauß periods can be found in the theses by Gao [9] and Nöcker [23]. In short, Gauß periods are certain sums of roots of unity. Intuitively, if ζ_r is an r-th root of unity in $\bar{\mathbb{F}}_3$, then $\mathbb{F}_3(\zeta_r)$ will have extension degree dividing $\phi(r)$. Under certain conditions specific sums of ζ_r and its conjugates will provide normal bases for $\mathbb{F}_{3^{\phi(r)}}$ and its subfields. For the general theory we refer to the theses by Gao and Nöcker, we limit ourselves to ground field $\mathbb{F}_3$ with prime extension degree and r a prime. For ease of reference we use Nöcker's notation as much as possible.

Let $q = 3$, let m be the desired extension degree. Let r be a prime such that $\phi(r) = r - 1 = mk$ for some integer k, called the type of the normal basis. Let $\mathcal{K}$ be a subgroup of $\mathbb{Z}_r^*$ of order k. Note that for a prime r this subgroup is unique. A Gauß period of type $(m, \mathcal{K})$ (or of type-k for short) is then defined as

$$\alpha = \sum_{a \in \mathcal{K}} \zeta_r^a \ .$$

In order for α to be normal, it is required that q and $\mathcal{K}$ together span the multiplicative group $\mathbb{Z}_r^*$. It is possible to determine the minimal polynomial of α by factoring $\Phi_r(X)$ over $\mathbb{F}_3$, resulting in a minimal polynomial of an r-th root of unity ζ_r and hence in a polynomial basis for $\mathbb{F}_r(\zeta_r)$. In this extension field the minimal polynomial of α can be computed by $\prod_{i=0}^{m-1}(X - \alpha^{3^{ki}})$. Once the minimal polynomial f has been computed it is easy to compute the multiplication matrix M. There is an alternative, more direct approach, constructing the matrix M based on the structure of Gauß periods.

Since in pairing based cryptography one selects m to be prime, as demonstrated by Table 1, a type-one normal basis or, equivalently for odd characteristic an optimal normal basis, is never available. This is unfortunate since type-one normal bases offer the highest level of performance due to the sparsity of their multiplication matrices.

However, we can construct a type-two normal basis for our prime values of m if $r = 2m + 1$ is also prime. In this case $(q, \mathcal{K})$ will always span $\mathbb{Z}_r^*$, since m must divide the order of q modulo r by virtue of m being prime and $3^k < mk$ for $m > 3$. Hence the Gauß period α will be normal in this case.

As an historical note, such values of m are termed Sophie Germain primes after the mathematician who, in 1825, proved that Fermat's Last Theorem is true for prime values of m when $(2 \cdot m) + 1$ is also prime. The number of these primes less than some value N is conjectured to be

$$2C_{TP} \int_2^N \frac{dx}{\log x \log(2x+1)} \sim \frac{2C_{TP}N}{(\log N)^2}$$

where C_{TP} is the twin prime constant. The number of these specific forms of m is clearly less than in the unrestricted case, but Table 2 shows that there are sufficiently many of a cryptographically interesting size that this should not be a problem. However, of the currently recommended parameterisations for pairing based cryptography only one field size, that where $m = 239$, yields a type-two normal basis.

To conclude, consider a small example where $m = 3$ that produces a usable type-two normal basis since $r = (2 \cdot 3) + 1 = 7$ is prime. We find a normal polynomial to define our basis in $\mathbb{F}_{3^3}$ to be

$$x^3 + x^2 + x + 2,$$

m	r	m	r	m	r
83	167	281	563	653	1307
89	179	293	587	659	1319
113	227	359	719	683	1367
131	263	419	839	719	1439
173	347	431	863	743	1487
179	359	443	887	761	1523
191	383	491	983	809	1619
233	467	509	1019	911	1823
239	479	593	1187	953	1907
251	503	641	1283		

Table 2. A table showing values of $80 < m < 1000$ where m and $r = (2 \cdot m) + 1$ are both prime, allowing a type-two normal basis to be constructed.

Field	Curve	Order	MOV security
$\mathbb{F}_{3^{89}}$	$Y^2 = X^3 - X + 1$	$(3^{89} - 3^{45} + 1)/C_1$	846
$\mathbb{F}_{3^{131}}$	$Y^2 = X^3 - X + 1$	$(3^{131} + 3^{66} + 1)/C_2$	1245
$\mathbb{F}_{3^{173}}$	$Y^2 = X^3 - X + 1$	$(3^{173} - 3^{87} + 1)/C_3$	1645
$\mathbb{F}_{3^{179}}$	$Y^2 = X^3 - X - 1$	$(3^{179} - 3^{90} + 1)/C_4$	1702
$\mathbb{F}_{3^{251}}$	$Y^2 = X^3 - X + 1$	$(3^{251} + 3^{126} + 1)/C_5$	2386
$C_1 = 15991171$			
$C_2 = 56844236505445613531121264 31$			
$C_3 = 16420688749$			
$C_4 = 2592169385514147730111519261$			
$C_5 = 923566965086821187474224034608441725745012784 77$			

Table 3. A table of normal basis friendly field and curve parameterisations.

and hence calculate the multiplication matrix as

$$M = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 2 & 1 \\ 1 & 1 & 1 \end{pmatrix}.$$

Notice that the first and second rows, the second being notable since $M_{2,2} = 2$, both have two non-zero entries while the third row has three. This is also a feature of larger fields constructed in this way although this, and the sparsity of the matrix are not apparent due to the small size of the example.

4 Suitable Curve Parameterisations

The list of valid selections of m in Table 2 is only part of the story as regards building a working parameterisation. Having specified the field $\mathbb{F}_q$, we must select a curve E over this field that is both suitable in terms of structure and security against attack. The list of potential m values in Table 2 was sparse compared to the general case since we put constraints on the acceptable values: suitable curve parameters are even sparser due to these constraints.

In a rough sense, the parameters in Table 1 were found by searching curves of the form $Y^2 = X^3 - X \pm 1$ for ones with large prime order, accommodating small cofactors, and with appropriate MOV security properties. Due to the extra constraints on m, normal basis friendly parameterisations are difficult to find. However, Table 3 shows the result of a limited search for such parameters.

Most notable are the curves for $m = 173$ and $m = 179$ which appear to offer a good balance between performance and security for the types of constrained environments we are interested in. The clear downside to these curves are the unattractively large cofactors which present some security issues relating to the possibility for small subgroup attacks, in addition to the performance impact of using large fields that only yield relatively small elliptic curve groups. Analysis of this sort of issue in the context of pairings and the search for further normal basis friendly parameters is an ongoing task, but currently seems a problem with respect to practical application.

In this paper we deal only with fields which allow a type-two normal basis, partly for brevity and partly because higher types yield lower performance. One way to alleviate this difficulty of curve parameterisation is to utilise the current recommendations from Table 1 and deal with higher complexity types. For example, one might wish to use $m = 163$ where $r = (4 \cdot 163) + 1 = 653$ is prime and hence a type-four normal basis can be constructed. We defer consideration of this approach for further work.

5 Arithmetic in Characteristic Three

In a physical sense, we follow other work [11] and represent a polynomial a as two bit-vectors a^H and a^L. If we let a_i^H and a_i^L denote bit i of a^H and a^L respectively, the vectors a^H and a^L are constructed from a such that for all i $a_i^H = a_i \text{ div } 2$ and $a_i^L = a_i \bmod 2$. That is, a^H and a^L are a bit-sliced representation of the coefficients of a where a^H and a^L hold the high and low bits of a given coefficient. Note that aside from where it matters, we abstract this representation away and simply assume that operations are applied to suitable pairs of bit vectors.

5.1 Addition, Subtraction and Multiplication of Coefficients

Component-wise operations on field elements in polynomial and normal bases are the same since they simply operate on pairs of coefficients, reducing the result so it lies in $\mathbb{F}_3$. Given our bit-sliced representation of polynomials, we can construct component-wise addition, subtraction and multiplication using simple logical operations. For example, component-wise addition $r_i = a_i + b_i$ of two polynomials a and b can be specified using the following logical operations

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

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 vectors a^H and a^L over and can therefore be implemented by the same function as addition. For normal basis arithmetic, we also require a component-wise multiplication $r_i = a_i \cdot b_i$. This can be performed using similarly inexpensive logical operations

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

On a given computer with word-size w, we hold the bit-vectors a^H and a^L that represent a as two word-vectors of length $\lceil m/w \rceil$ and hence apply logical operations in parallel to w coefficients at a time. This is convenient since not only are logical operations cheap to process, the use of large SIMD registers to accelerate execution is also very easy.

5.2 Cubing and Cube Roots

In characteristic three, cubing is a linear operation in the same way squaring is linear in characteristic two. Therefore, when working in characteristic three cubing is an important operation since curve and pairing arithmetic is specifically manipulated to utilise cubing over more costly multiplication. In addition, the cube root operation is important in the Duursma-Lee pairing arithmetic if pre-computation is avoided.

Our reason for considering normal bases in the first place was the efficiency of cube and cube root operations 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 software and even more so in a hardware circuit, where they reduce to wired permutation of bits with no actual computational overhead.

Since the field representation is both compact and bit-oriented, when using a polynomial basis the cube operation can be implemented using table look-up in an analogous way to the coefficient thinning method in characteristic two. Although this requires a subsequent reduction operation, it is an order of magnitude less expensive than multiplication. Taking cube roots is a little more awkward but can be accelerated using a trick involving a small amount of pre-computation. Recall that one can write an element $\hat{a}$ in a polynomial basis as

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

By expanding this summation and extracting cube roots from appropriate summands, we find that

$$\begin{aligned}
\sqrt[3]{\hat{a}} &= \sqrt[3]{\textstyle\sum_{i=0}^{m-1} \hat{a}_i \cdot \alpha^i} \\
&= \sqrt[3]{\textstyle\sum_{i=0}^{\lceil \frac{m}{3} \rceil - 1} (\hat{a}_{3i} \cdot \alpha^{3i}) + (\hat{a}_{3i+1} \cdot \alpha^{3i+1}) + (\hat{a}_{3i+2} \cdot \alpha^{3i+2})} \\
&= \textstyle\sum_{i=0}^{\lceil \frac{m}{3} \rceil - 1} (\hat{a}_{3i} \cdot \alpha^{i}) + \alpha^{1/3} \cdot (\hat{a}_{3i+1} \cdot \alpha^{i}) + \alpha^{2/3} \cdot (\hat{a}_{3i+2} \cdot \alpha^{i}).
\end{aligned}$$

That is, if we pre-compute the values of $\alpha^{1/3}$ and $\alpha^{2/3}$ the cube root operation is reduced to two multiplications, which can be further optimised since the operands will always be a third of the length of a full element, and two additions.

5.3 Multiplication

Efficient multiplication in finite fields of characteristic two is a well studied topic. Methods in characteristic three are less mature but since our representation of field elements is bit-oriented, we open the possibility of converting several methods from characteristic two.

Using a polynomial basis in software, one can easily construct a characteristic three version of the comb method [20]. After extensive experimentation however, a Karatsuba-Ofman style approach seems the fastest choice. We utilise the two-way splitting of conventional Karatsuba-Ofman [18], as well as three-way splitting proposed by Bailey and Paar [1], to reduce the operands to word sized objects where we then use standard polynomial multiplication. Hardware polynomial basis multipliers have also been developed [24], the most efficient being that of Bertoni *et al.* [3].

As mentioned previously, multiplication of normal basis field elements is dictated by a matrix which essential encodes how reduction takes place. Given such a matrix M, constructed from the normal polynomial f, we can generate coefficients of the result $\bar{c} = \bar{a} \cdot \bar{b}$ 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.

In characteristic two, hardware normal basis multipliers have seen increasingly high performance starting with the implementation of Wang *et al.* [31] who present results for a Massey-Omura based design. Since then, specific optimisations have yielded fast circuits [12, 19, 26] for specific classes of field.

Algorithms for efficient software implementation of this operation in characteristic two have been presented by Reyhani-Masoleh and Hasan [25] and also Ning and Yin [22]. For working in characteristic three, we adopt the later method since the operations map naturally onto our bit-sliced representation of field elements. Specifically, we adapt the Algorithm 3 of Ning and Yin [22] to suit our purposes and to some extent adopt their terminology by presenting the algorithm in C style pseudo-code. Note that in this pseudo-code, both addition and multiplication operations are assumed to be component-wise modulo three.

The basic method, shown in Algorithm 3, revolves around the pre-computed arrays A and B where each entry in the $2m$ sized arrays is a w-bit sized word. Although this method of pre-computation is perhaps more costly than that of Reyhani-Masoleh and Hasan [25], it offers a major performance benefit when considering the memory characteristics of accesses to A and B. For example, since the rotated words are held sequentially, using this method provides good cache locality. Furthermore, since the method deals only with word sized objects, rather than full rotated field elements, the pre-computation itself is faster.

The algorithm also relies on the multiplication matrix M, described in the construction of the normal basis. However, since the matrix is sparse, we only hold the non-zero values in arrays t_0, t_1 and t_3 where $t_i[j]$ holds the i-th non-zero value of row j for $0 < i \leq 2$ and $0 < j < m$. Note that since $t_3[0]$ is never a valid non-zero row index, we treat this as a special case before entry to the main loop. Also note that we neglect to check for the case when $M_{i,j} = 2$ which will occur once in the matrix. Since this event occurs only once, at a position where $t_3[j]$ is again invalid, we introduce an extra addition and treat $2 \cdot \bar{b}_{k+j}$ as $\bar{b}_{k+j} + \bar{b}_{k+j}$ so as to remove the test otherwise required on each iteration of the loop.

5.4 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. With characteristic three fields in polynomial basis, we can use a simple translation of the standard binary Euclidean algorithm to invert elements. Although still slow in comparison to other operations, this translation is made somewhat more natural thanks to the bit-oriented form of the field representation.

Inversion of elements held in a normal basis is far more costly. Since one can not use Itoh-Tsujii type methods [15] to reduce the cost thanks to the form of m, the best way to invert an elements seems to be simply powering it

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

This should be implemented using a ternary expansion of the exponent since cubing operations are so inexpensive.

Algorithm 3: A software algorithm for type-two normal basis multiplication in characteristic three.

Input : Field elements $\bar{a}$ and $\bar{b}$

Output : The element $\bar{c} = \bar{a} \cdot \bar{b} \pmod{f}$

Pre-computation

`A[i]` = `A[i+m]` = $(\bar{a}_i, \bar{a}_{i+1}, ..., \bar{a}_{i+w-1})$
`B[i]` = `B[i+m]` = $(\bar{b}_i, \bar{b}_{i+1}, ..., \bar{b}_{i+w-1})$

Multiplication

```
for( k = 0; k < m; k += w )
{
  t = A[ 0 ] * ( B[ t1[0] ] +
                 B[ t2[0] ] );

  for( i = 1; i < m; i++ )
  {
    t += A[ i ] * ( B[ t1[i] ] +
                    B[ t2[i] ] +
                    B[ t3[i] ] );
  }

  C[ k ] = t;

  A += w;
  B += w;
}
```

6 Software Implementation

In order to provide some concrete idea of the practical cost of the presented software based methods, we implemented the proposed field arithmetic and pairing algorithms. This is clearly of interest since a comparison between polynomial and normal basis implementations will effectively determine the best method for realising high performance software only pairings on memory constrained devices. We used a GCC 3.3 compiler suite to build our implementation and ran timing experiments on a Linux based PC incorporating a 2.80 GHz Intel Pentium 4 processor. The entire system was constructed in C++. We accept that further performance improvements could be made through aggressive profiling and optimisation but are confident our results are representative of the underlying algorithms and allow a comparison between them.

Table 4 shows timings from our implementation. Note that both normal and polynomial bases timings are included and that $+PC$ and $-PC$ represent

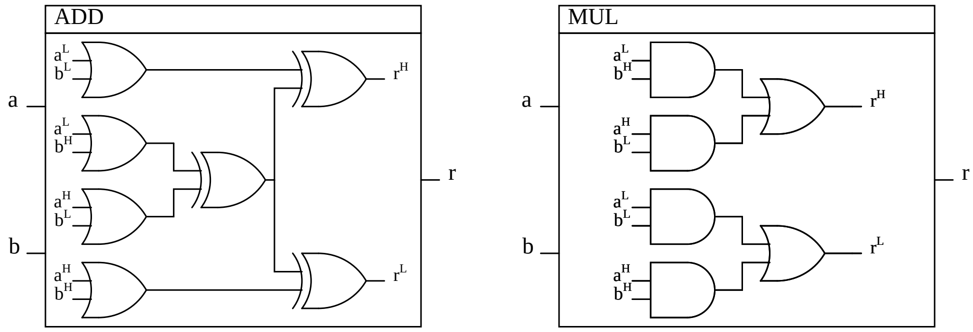
	Polynomial	Normal
Arithmetic in $\mathbb{F}_{3^{239}}$		
Add	$0.59\mu s$	$0.56\mu s$
Multiply	$19.72\mu s$	$52.16\mu s$
Square	$19.24\mu s$	$50.04\mu s$
Cube	$1.36\mu s$	$0.53\mu s$
Cube Root	$16.50\mu s$	$0.51\mu s$
Invert	$136.34\mu s$	$12156.00\mu s$
Arithmetic in $\mathbb{F}_{3^{239\cdot 6}}$		
Add	$1.20\mu s$	$1.15\mu s$
Multiply	$367.70\mu s$	$987.45\mu s$
Square	$344.90\mu s$	$984.14\mu s$
Cube	$5.50\mu s$	$1.32\mu s$
Invert	$819.01\mu s$	$14920.10\mu s$
Pairings		
BKLS	$93.76ms$	$449.30ms$
Duursma-Lee -PC	$78.16ms$	$171.60ms$
Duursma-Lee +PC	$66.46ms$	$170.10ms$

Table 4. Timings for field arithmetic in $\mathbb{F}_{3^{239}}$, $\mathbb{F}_{3^{239\cdot 6}}$ and Duursma-Lee based pairings using polynomial and normal bases.

results with and without pre-computation as described in Section 2. Working in a polynomial basis is clearly the faster of the two methods, even though cube and cube root operations are far quicker in the normal basis arithmetic: using pre-computation has the effect of accelerating the pairing using the polynomial basis as a result. Over all, the polynomial basis pairing is between two and three times as fast as the normal basis alternative, even when no pre-computation is used. This is basically a product of the differences in multiplication speed since this, rather than the cube and cube root operations, is the dominant cost in computation.

A secondary performance problem is the exceptionally high cost of inversion in normal bases. Even considering the conversion [17] before and after inversion in a polynomial basis, this seems a very expensive operation. However, in practice it is possible to avoid inversions in many protocols since one ordinarily needs only verify equality of pairing outputs belonging to the group $\mathbb{F}^*_{q^6}/\mathbb{F}^*_{q^3}$. Using the methods of [10] one can perform an equality check with just two $\mathbb{F}_{q^3}$ multiplications.

With regard to other aspects of protocols, particularly post-pairing exponentiation in $\mathbb{F}_{q^6}$, it is desirable to be able to perform inversions [10]. In this case it makes sense to map an element in a normal basis representation to a polynomial representation, perform the inversion and then map back again. In general this isomorphism will cost approximately $m^2/3$ multiplications in $\mathbb{F}_3$, and is thus



Addition. Multiplication.

Fig. 1. Circuit diagrams for component-wise addition and multiplication in $\mathbb{F}_3$. Note that at a higher level, we abstract the internal bit-sliced operations to make things clearer.

quite expensive, but based on the performance comparison as detailed in Table 4, this is clearly a reasonable option.

7 Hardware Implementation

Consider the same example from Section 3 where $m = 3$. Using the matrix M, and the multiplication equation from Section 5.3, we take the basic ADD and MUL circuits in Figure 1 and construct a fully parallel multiplier as shown in Figure 2. Note that we are able to reduce the number of required addition circuits by reuse of previously computed results.

Multiplication circuits for larger fields can be constructed by considering the specific example in Figure 2 as parallel instances of a circuit that generates a single result coefficient. This circuit can be split into three phases as demonstrated by Figure 3: a generation phase that produces m sums of coefficients in b; a multiplication phase that multiplies these sums with coefficients of a; and a accumulation phase that adds all the multiplication results together to get the final result coefficient. Although very basic, this design is very regular and fairly flexible in the sense that one can build a fully or partially parallel device from the same basic building blocks.

7.1 Cost Analysis

In calculating the cost of multipliers of this type, we use A_A, A_O and A_X to denote area required for logical and, or and exclusive-or gates and T_A, T_O and T_X to denote the time delay introduced by those gates. Using this notation, we find that our ADD and SUB circuits both require an area of $4A_O + 3A_X$ gates to build and have a delay of $T_O + 2T_X$. The MUL gate on the other hand

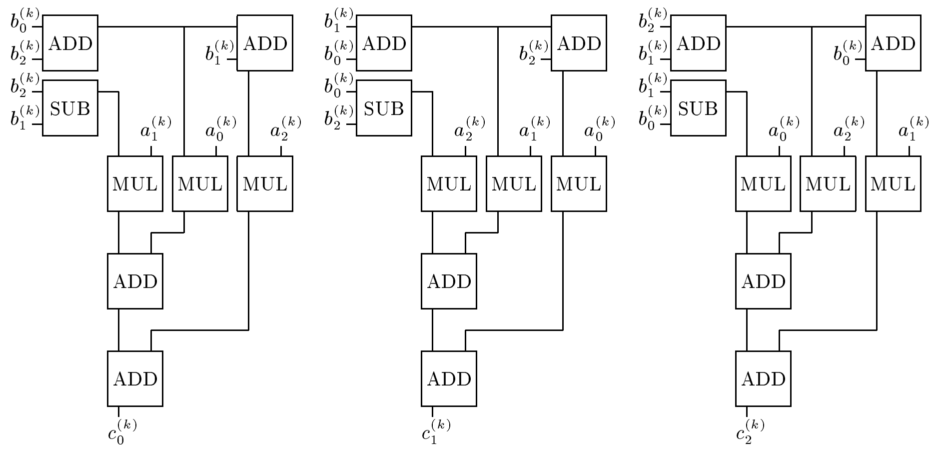


Fig. 2. A normal basis multiplier for the field F_{3^3}.

requires an area of $4A_A + 2A_O$ gates to build and has a delay of $T_A + T_O$. From the abstract multiplier in Figure 3, we use the costs associated with these basis circuits to derive a loose upper bound for both the total number of gates required and the overall delay.

The generate phase is required to produce m sums of the input coefficients. As a result of the way the matrix M is constructed for our basis, each of these sums contains a maximum of three summands. Exceptions to this rule are the first sum and the sum containing the term $M_{i,j} = 2$, both of which contain two summands and that in the latter case is calculated using a SUB circuit rather than an ADD. Therefore, an upper bound for the area of gates in the generate phase is $(3m - 2) \cdot (4A_O + 3A_X)$. The gate delay is fixed by the longest path, i.e. that with three summands, and is hence $3 \cdot (T_O + 2T_X)$. Note that in reality, less gates are required for this phase since partial sums can often be read from previous ones. This is demonstrated in Figure 2 where, for example, the result $b_0^{(k)} + b_1^{(k)} + b_2^{(k)}$ is constructed by adding $b_1^{(k)}$ to the previously computed value $b_0^{(k)} + b_2^{(k)}$.

The multiply phase requires m parallel MUL circuits and hence an area of $m \cdot (4A_A + 2A_O)$ gates with a delay of $T_A + T_O$. The accumulate phase sums the m outputs from the multiply phase. Using a binary tree for this task, the accumulate phase results in a structure of height $h = \lceil \log_2(m) \rceil$. Such a structure can be built using $(m - 1)$ interior nodes, each composed of an ADD circuit, meaning a total area of $(m - 1) \cdot (4A_O + 3A_X)$ gates that impose a delay of $h \cdot (T_O + 2T_X)$.

To construct a fully parallel multiplier, we need to place m of these phases in order to generate m coefficients of the result. Using the previous cost analysis as a platform, we find that an upper bound for the number of gates required for

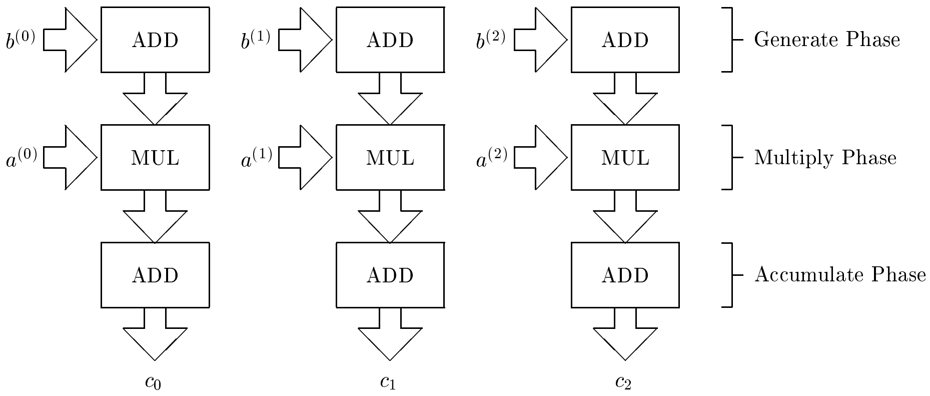


Fig. 3. Abstract multiplier showing the phases of operation.

each result coefficient generator can be stated as

$$(4m - 3) \cdot (4A_O + 3A_X) + m \cdot (4A_A + 2A_O),$$

while the maximum delay imposed by those gates is

$$(3 + \lceil \log_2(m) \rceil) \cdot (T_O + 2T_X) + (T_A + T_O).$$

Since we intend to place many of these units in parallel, we will see an m-fold increase in gates but note that as long as the multiplier is fully parallel, the delay to produce the result remains the same. Of course, one can make a trade-off between size and speed by placing only $n < m$ parallel result coefficient generators and using them iteratively to produce the final result n coefficients at a time.

Putting all of this into context, a fully parallel multiplier for the case where $m = 239$ would require

$$228484A_A + 1025310A_O + 683301A_X$$

gates, i.e. almost two million gates, and the delay between results being produced would be

$$T_A + 12T_O + 22T_X.$$

Clearly this is far too large a device to deploy on a smart-card so one might consider only placing a single result coefficient generator, reducing the gate count to a manageable

$$956A_A + 4290A_O + 2859A_X,$$

that is around eight thousand gates, although delaying the result by a factor of m. Given the cost of, for example, Montgomery multipliers currently housed on smart-cards eight thousand gates seems like a fairly modest total. However this estimate is clearly very rough, focusing on the multiplier only and ignoring issues such as control logic.

7.2 Performance Analysis

In order to minimise the amount of logic required, we assume a worst case in terms of environmental constraints: there is only room on the device for a single result coefficient generation circuit. Given this architecture and a modest clock speed, it seems possible to generate a single coefficient of the result every clock cycle for reasonable sizes of m, meaning we expect to complete a multiplication every m cycles. Although this neglects the cost of, for example, loading and storing values to and from memory before and after completion of the multiplication, it seems pessimistic enough to allow reasonable ball-park analysis.

If we assume that efficient methods for field arithmetic and removal of the final exponentiation are used [10], the Duursma-Lee method described in Section 2 can be computed using $14m$ multiplications in $\mathbb{F}_{3^m}$ and a handful of auxiliary operations such as addition. Since the auxiliary operations, including the cube and cube root stages, are inexpensive in comparison the multiplication the total time required to process the algorithm is dominated by the said $14m$ multiplications. Given a modest smart-card clock speed of 8MHz and our multiplier for $m = 239$, we expect to process a single multiplication in 239 cycles. Ignoring the cost of any auxiliary operations, the whole pairing algorithm could complete in well under a second. Even including these auxiliary operations, it seems reasonable that the operation should complete close to this mark.

Clearly this whole argument depends on the operation of the multiplier in context: it is meaningless to quote clock speeds and cycle times for the multiplier without a overarching design. However, the fact that a low cost multiplier circuit can accommodate the dominant computational load of the pairing in a useful time period is a strong step towards this goal. This is especially true since in constrained environments, $m = 239$ is probably overkill in terms of security and time constraints are more elastic than in interactive applications: a one or two second wait at a smart-card enabled ATM machine is unlikely to be apparent to the user.

8 Conclusions

In this paper we presented methods for constructing and using normal basis arithmetic in characteristic three and then applied it to the context of pairing based cryptography. We showed that although fast methods for normal basis multiplication in software can be constructed, they are still too slow when compared to a polynomial basis in the context of computing the pairing. However, this drawback is eliminated when considering hardware implementation where acceleration devices can be high performance while maintaining a low cost in terms of area and time. In this context, normal bases offer a fast way to perform pairing computation while removing the need for any pre-computation that would hinder implementation using a polynomial basis. The applications for such as design are clear: constrained devices such a smart-cards which were previously thought to possess too little computational power can feasible implement identity based cryptography.

However, in both hardware and software, we noted that depending on the protocol that uses the pairing value, the use of normal bases could be made unattractive due to the high cost of inversion in the base field and hence the difficulty of efficiently performing further arithmetic on said value. We presented a method to reduce this cost but stress that this problem, and the more important issues that surround construction of curves that allow type-two normal bases, could preclude their use as envisaged by Duursma and Lee all together. Hence, even though we improve significantly on previous work, there are still several areas that require further investigation.

Use a polynomial basis In this paper we focused on the use of normal basis arithmetic with the goal of constructing a hardware accelerator for the Duursma-Lee algorithm. With this as the emphasis, we ignored the possibility of using a polynomial basis and computing cube roots on the fly with the method from Section 5.2. Given the cost of this method is less than a general multiply, it seems feasible that an adequate accelerator could be constructed by ignoring normal basis arithmetic altogether. We hope to investigate this possibility in further work.

Nöcker style multiplication In his thesis [23], Nöcker introduces a method for performing normal basis multiplication by doing a fast translation to a polynomial basis, doing the multiplication and then translating back again. If this can be achieved quicker than our current methods, it could be attractive. In hardware, it could be an especially good idea since it could allow accelerated support for both polynomial *and* normal basis arithmetic using only a single, conventional polynomial multiplier.

Normal bases with $t > 2$ As outlined in Section 4, we deal only with type-two normal bases even though higher complexity types can be useful when considering the problem of curve parameterisation. This trade-off remains an open problem that is in some sense application specific, i.e. the system designer must match the cost of an architecture against the security constraints. However, it seems an interesting problem to investigate if there are better methods than using Gaußian normal bases for more complex types, both in terms of construction and arithmetic.

Architectural optimisations In the same way that architectures for normal and optimal normal basis multipliers in characteristic two have matured, architectural optimisation of our rather basic design seems inevitable. For example, some form of pipelining could clearly be useful and area minimisations are presumably possible. Although we prototyped our design in Verilog, we hope to address this area by constructing a complete accelerator design and implementing it on an FPGA. This work is sure to encompass a dual goal to the one considered here: as well as constrained devices, it is attractive to design very high performance accelerator devices for server side applications. By taking advantage of lifting constraints of area and clock speed, the performance of such applications can be improved in the same way that SSL servers are accelerated using dedicated RSA hardware.

Acknowledgements

We are deeply indebted to Fré Vercauteren for pointing out the cube root method in Section 5.2 and letting us know when we had things back to front.

References

1. D. Bailey and C. Paar. Efficient Arithmetic in Finite Field Extensions with Application in Elliptic Curve Cryptography. In *Journal of Cryptology*, **14** (3), 153–176, 2001.
2. P. Barreto, H. Kim, B. Lynn and M. Scott. Efficient Algorithms for Pairing-Based Cryptosystems. In *Advances in Cryptology (CRYPTO)*, Springer LNCS 2442, 354–368, 2002.
3. 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.
4. D. Boneh and M. Franklin. Identity-Based Encryption from the Weil Pairing. In *SIAM Journal on Computing*, Volume 32, no. 3, 586–615, 2003.
5. D. Boneh, B. Lynn and H. Shacham. Short signatures from the Weil Pairing. In *Advances in Cryptology (ASIACRYPT)*, Springer LNCS 2248, 514–532, 2001.
6. I. Duursma and H. Lee. Tate Pairing Implementation for Hyperelliptic Curves $y^2 = x^p - x + d$. In *Advances in Cryptology (ASIACRYPT)*, Springer LNCS 2894, 111–123, 2003.
7. G. Frey and H. Ruck. A Remark Concerning m-Divisibility and the Discrete Logarithm Problem in the Divisor Class Group of Curves. In *Mathematics of Computation*, **62**, 865–874, 1994.
8. S. Galbraith, K. Harrison and D. Soldera. Implementing the Tate pairing. In *Procedings of ANTS V*, Springer LNCS 2369, 324–337, 2002.
9. S. Gao. Normal Bases over Finite Fields. PhD Thesis, Waterloo University, 1993.
10. R. Granger, D. Page and M. Stam. On Small Characteristic Algebraic Tori in Pairing-Based Cryptography. Cryptology ePrint Archive, Report 2004/132. Available from `http://eprint.iacr.org/2004/132`.
11. 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.

12. M.A. Hassan, M.Z. Wang and V.K. Bhargava. A Modified Massey-Omura Parallel Multiplier for a Class for Finite Fields. In *IEEE Transactions on Computers*, **42** (10), 1278–1280, 1993.
13. F. Hess. Efficient Identity based Signature Schemes based on Pairings. In *Selected Areas in Cryptography (SAC)*, Springer LNCS 2595, 310–324, 2003.
14. IEEE P1363. Standard Specifications for Public Key Cryptography. IEEE Standards Department, 1999.
15. 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.
16. A. Joux. A One Round Protocol for Tripartite Diffie-Hellman. In *Proc. of ANTS IV*, Springer LNCS 1838, 385–394, 2000.
17. B.S. Kaliski Jr. and Y.L. Yin. Storage Efficient Finite Field Basis Conversion. In *Selected Areas in Cryptography (SAC)*, Springer-Verlag LNCS 1556, 81–93, 1999.
18. A. Karatsuba and Y. Ofman. Multiplication of Many-Digital Numbers by Automatic Computers. In *Doklady Akad. Nauk SSSR* **145**, 293–294, 1962. Translation in *Physics-Doklady* **7**, 595–596, 1963.
19. Ç.K. Koç and B. Sunar. Low-Complexity Bit-Parallel Canonical and Normal Basis Multipliers for a Class for Finite Fields. In *IEEE Transactions on Computers*, **47** (3), 353–356, 1998.
20. J. López and R. Dahab. High Speed Software Multiplication in $\mathbb{F}_{2^m}$. In *Progress in Cryptography (INDOCRYPT)*, Springer-Verlag LNCS 1977, 203–212, 2000.
21. 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.
22. P. Ning and Y.L. Yin. Efficient Software Implementation for Finite Field Multiplication in Normal Basis. In *Information and Communications Security (ICICS)*, Springer-Verlag LNCS 2229, 177–188, 2001.
23. M. Nöcker. Data Structures for Parallel Exponentiation in Finite Fields. PhD Thesis, Universität Paderborn, 2001.
24. 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.
25. A. Reyhani-Masoleh and M.A. Hasan. Fast Normal Basis Multiplication Using General Purpose Processors. In *Selected Areas in Cryptography (SAC)*, Springer LNCS 2259, 230–244, 2001.
26. A. Reyhani-Masoleh and M.A. Hassan. A New Construction of Massey-Omura Parallel Multiplier over $GF(2^m)$. In *IEEE Transactions on Computers*, **51** (5), 511–520, 2002.
27. R. Sakai, K. Ohgishi and M. Kasahara. Cryptosystems Based on Pairings. In *Symposium on Cryptography and Information Security (SCIS)*, 2000.
28. M. Scott and P. Barreto. Compressed Pairings. Cryptology ePrint Archive, Report 2004/032. Available from `http://eprint.iacr.org/2004/032`.
29. A. Shamir. Identity-Based Cryptosystems and Signature Schemes. In *Advances in Cryptology (CRYPTO)*, Springer LNCS 196, 47–53, 1985.
30. J. Silverman. The Arithmetic of Elliptic Curves. Springer-Verlag GTM 106, 1986.
31. C.C. Wang, T.K. Truong, H.M. Shao, L.J. Deutsch, J.K. Omura and I.S. Reed. VLSI Architectures for Computing Multiplications and Inverses in $GF(2^m)$. In *IEEE Transactions on Computers*, **34** (8), 709–716, 1985.