Hardware Implementation of Finite Fields of Characteristic Three

D. Page and N.P. Smart

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

Abstract. In this paper we examine a number of ways of implementing characteristic three arithmetic in hardware. While this type of arithmetic is not traditionally used in cryptographic systems, recent advances in Tate and Weil pairing based cryptosystems show that it is potentially valuable. We examine a hardware oriented representation of the field elements, comparing the resulting algorithms for field addition and multiplication operations, and show that characteristic three arithmetic need not significantly under-perform comparable characteristic two alternatives.

1 Introduction

There has been a recent increase in research activity surrounding cryptosystems based on the Tate and Weil pairings. Identity based encryption schemes [6] and signature algorithms [11,16,17] as well as general signature algorithms [7] have been developed and published, all of which utilise pairing based operations. Additionally, extensions to higher genus curves have been fully explored [8]. Pairing based cryptosystems were traditionally thought to be weak when it was shown [13] that the discrete logarithm problem in supersingular curves was reducible to that in a finite field using the Weil pairing. However, this view changed when Joux [12] presented a simple tripartite Diffie-Hellman protocol based on the Weil pairing on supersingular curves which, in part, rekindled interest in this area.

Although there is little discussion about implementation, it was noted by Galbraith [8] that in terms of bandwidth efficiency, it is more efficient to use elliptic curves in characteristic three for systems based on the Weil or Tate pairing. This notion contradicts conventional advice when implementing elliptic curves, which generally suggests using fields of either large prime characteristic or characteristic two. The use of such fields is generally based on the assumption that arithmetic in characteristic three is much slower than the given alternatives and has resulted in a gap in literature surrounding the topic.

B.S. Kaliski Jr. et al. (Eds.): CHES 2002, LNCS 2523, pp. 529–539, 2003.

Since the efficient hardware implementation of elliptic curves arithmetic in characteristic three is potentially of value to the expanding list of systems which use the Weil or Tate pairing, we will fill this gap in this paper. The purpose of this work is to facilitate the use of characteristic three arithmetic in pairing based cryptosystems, and hence reap the advantages of doing so, without imposing the performance overhead which may traditionally be expected. In Section 2 we discuss the Tate pairing and develop some parameters for the comparison of our techniques. We present a way of representing polynomials and performing arithmetic on them in Sections 3 and 4. Finally, we implement these arithmetic operations in field programmable hardware and present the performance results in Section 5.

2 Supersingular Elliptic Curves and the Tate Pairing

We let $\mathbb{G}$ denote a prime order subgroup of an elliptic curve E over the field $\mathbb{F}_q$, which for the moment we assume is a general finite field of arbitrary characteristic. Let the order of $\mathbb{G}$ be denoted by l and define α to be the smallest integer such that

$$l | q^\alpha - 1$$

In practical implementations we will require α to be small and so will usually take E to be a supersingular curve over $\mathbb{F}_q$. Let $\overline{\mathbb{G}}$ denote the group of points of order l of the elliptic curve E over the field $\mathbb{F}_{q^\alpha}$. While the group $\mathbb{G}$ is cyclic of order l, the group $\overline{\mathbb{G}}$ is a product of two cyclic subgroups of order l.

The bandwidth performance of the schemes based on the Weil pairing usually grow with $\log_2 q$ rather than with $\alpha \cdot \log_2 q$, hence it is better to try to minimise q. This leads us to consider fields of characteristic three, since they aid us in minimising the value of q and hence minimising the bandwidth. However, it is unclear as to whether this comes at the expense of a decrease in performance when compared against fields of characteristic two. In this paper we go some way to address this issue in hardware by performing a comparison of the field primitives. A comparison of the actual protocols we leave to a later publication.

In this paper we shall be interested in protocols which make use of the modified Tate pairing given by the map

$$\hat{t} : \mathbb{G} \times \mathbb{G} \to \mathbb{F}_{q^\alpha}^*,$$

which satisfies the following properties

1. Bilinearity:
   - $\hat{t}(P_1 + P_2, Q) = \hat{t}(P_1, Q) \cdot \hat{t}(P_2, Q)$.
   - $\hat{t}(P, Q_1 + Q_2) = \hat{t}(P, Q_1) \cdot \hat{t}(P, Q_2)$.
2. Non-degeneracy: There exists a $P \in \mathbb{G}$ such that $\hat{t}(P, P) \neq 1$.
3. Computable : One can compute $\hat{t}(P, Q)$ in polynomial time.

If we let ϕ denote a "distortion map", or group endomorphism which maps elements in $E[l]$ into linearly independent elements of $E[l]$, then we can define

the modified Tate pairing from the standard Tate pairing $t(P,Q)$ via the use of distortion maps

$$\hat{t}(P,Q) = t(P,\phi(Q))^{(q^\alpha-1)/l}$$

That the Tate pairing is efficiently computable follows from an unpublished, but much referenced, algorithm of Miller [14].

We wish to compute $\hat{t}(P,Q)$ where $P,Q \in \mathbb{G}$. This requires some operations to be performed in $\mathbb{F}_q$ and some to be performed in $\mathbb{F}_{q^\alpha}$, see [5] and [9]. The exact value of α depends on which supersingular curve is chosen. The optimal choices in each characteristic are given by the following table

Field	Curve	α
$\mathbb{F}_{2^p}$	$y^2+y=x^3+x$	4
$\mathbb{F}_{2^p}$	$y^2+y=x^3+x+1$	4
$\mathbb{F}_{3^p}$	$y^2=x^3-x+1$	6
$\mathbb{F}_{3^p}$	$y^2=x^3-x-1$	6
$\mathbb{F}_p$	$y^2=x^3+1$	2
$\mathbb{F}_p$	$y^2=x^3+x$	2

Notice, the value of α is bounded by four in characteristic two, by six in characteristic three and two for curves defined over large prime fields. The underlying security of the system is based both on the computational Diffie-Hellman problem in the subgroup of order l of $E(\mathbb{F}_q)$ (the so called ECDLP security) and on the computational Diffie-Hellman problem in the finite field $\mathbb{F}^*_{q^\alpha}$ (the so called MOV security). Note that the decision Diffie-Hellman problem on supersingular elliptic curves is easy due to the existence of the Weil and Tate pairings, as was first pointed out by Joux [12].

We therefore need to choose, assuming standard current security recommendations,

- $l \approx 2^{160}$
- $q^\alpha \approx 2^{1024}$

If we wish to deploy a system with security roughly equivalent to 1024-bit RSA or 160-bit ECC, then we are led to consider the following parameters in each characteristic

Field	Curve	ECDLP Security	MOV Security
$\mathbb{F}_{3^{97}}$	$y^2=x^3-x+1$	151	922
$\mathbb{F}_{2^{241}}$	$y^2+y=x^3+x+1$	241	964

We shall consider these parameters when describing our implementation of characteristic three arithmetic below, and the corresponding characteristic two implementation with which we compare it.

3 Polynomial Arithmetic Modulo Three

In order to improve on the expected performance of characteristic three arithmetic, we decided to use a novel representation of polynomials [10]. Each set of

polynomial coefficients is held as two values, which we shall denote w_1 and w_2. A given bit in w_1 is set if the corresponding coefficient of the polynomial is equal to one, while if the given bit in w_2 is set then the coefficient of the polynomial is equal to two. If both bits are clear then the coefficient is zero, while the case of both bits set is considered invalid.

Put more simply, w_1 holds the least significant bits of all coefficients in the polynomial while w_2 holds the most significant bits. This method of holding the coefficients is similar to the practice of bit-slicing which is often performed in software. By bit-slicing the high and low bits of each coefficient into separate values, we offer a much more effective way to perform arithmetic as well as a natural representation which is bit oriented in the same way that characteristic two arithmetic is commonly implemented. As an example of this representation, consider the trinomial $x^6 + x + 2$ which can be described as in Figure 1

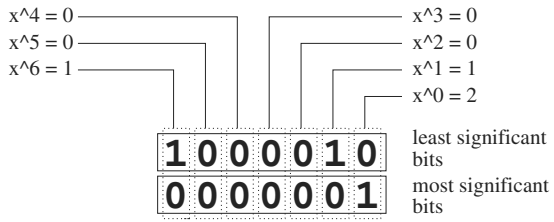

Fig. 1. Bit-sliced Representation

Note that as we are working in hardware and not tied to a word oriented design, where each coefficient occupies a number of bits which roughly equate to the word-size of a processor, this representation is far more compact than other methods. The size of w_1 and w_2 simply grow in length as the degree of the polynomial they represent grows.

3.1 Addition

Addition of polynomials is done on a per-value basis using seven logic operations. Consider the example which adds the polynomial represented by the values (a_1, a_2) to the polynomial (b_1, b_2), producing a result in (r_1, r_2). We can express the addition as a logic diagram, shown in Figure 2, or in the form of a simple pseudo-code program

```
t  = (a1 | b2) ^ (a2 | b1);
r1 = (a2 | b2) ^ t;
r2 = (a1 | b1) ^ t;
```

Note that negation and multiplication by two in this representation are particularly easy operations to implement since

$$2 \cdot (a_1, a_2) = -(a_1, a_2) = (a_2, a_1)$$

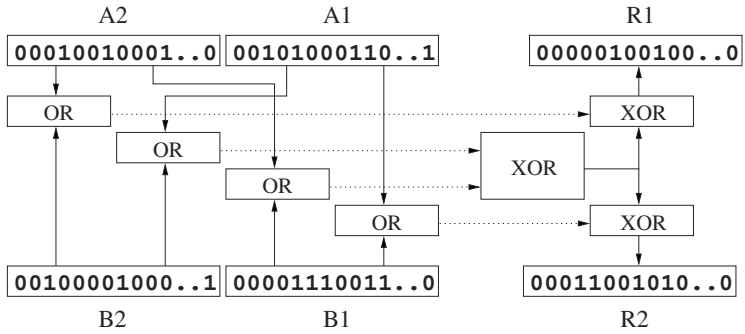

Fig. 2. Addition

3.2 Multiplication

A natural way to multiply elements in this representation is in a bit-serial manner. In this method we take two operands and perform a multiply by repeatedly shifting the multiplier down by one bit position and shifting the multiplicand up by one bit position. The multiplicand is then added or subtracted from the output value, on each iteration, depending on whether the least significant bit of the first or second word of the multiplier is set to one. This is possible due to the identity mentioned above which notes that the double operation is equivalent to the negation operation.

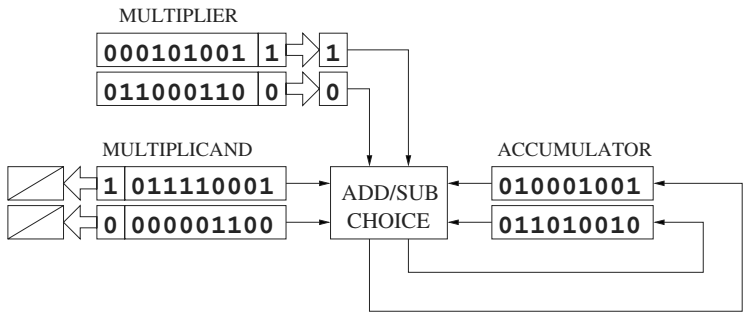

Fig. 3. Multiplication

The advantage of this full bit serial technique is that it requires less intermediate storage and is far more suited to a hardware implementation, using a basic iterated structure and only simple logic elements, i.e. no direct multiplier or adder circuitry is required. However, a major disadvantage of the full bit-serial multiplier is that an analogous cubing operation is only as fast as a general multiply, where as with other representation methods we can perform a more efficient cubing operation than a general multiply.

4 Implementation of Arithmetic in $\mathbb{F}_{3^{6p}}$

When considering pairing based cryptosystems, we are not only required to perform some operations in $\mathbb{F}_{3^p}$ but will also need to compute in the extension $\mathbb{F}_{3^{6p}}$. Since in applications p is a prime greater than five we can use the following representation of the finite field $\mathbb{F}_{3^{6p}}$

$$\mathbb{F}_{3^{6p}} = \mathbb{F}_{3^p}[\theta]/(\theta^6 + \theta + 2)$$

This provides a performance efficient reduction operation for multiplication. For example, consider the multiplication of two polynomials, a and b, in the field $\mathbb{F}_{3^{6p}}$ which we denote

$$a = a_5\theta^5 + a_4\theta^4 + a_3\theta^3 + a_2\theta^2 + a_1\theta + a_0$$

and

$$b = b_5\theta^5 + b_4\theta^4 + b_3\theta^3 + b_2\theta^2 + b_1\theta + b_0$$

Firstly, we multiply the two polynomials using a school-book method to produce a degree ten resulting polynomial r. We can then perform reduction of r, with respect to the irreducible trinomial $\theta^6 + \theta + 2$, using the circuitry as in Figure 4 since the multiplication results in

$$\begin{aligned} a \cdot b = r &= r_{10}\theta^{10} + r_9\theta^9 + \cdots + r_2\theta^2 + r_1\theta + r_0 \\ &= s_5\theta^5 + s_4\theta^4 + s_3\theta^3 + s_2\theta^2 + s_1\theta + s_0 \end{aligned}$$

and we know that $\theta^6 = 2\theta + 1$, so

$$\begin{aligned} s_0 &= r_0 + r_6 \\ s_1 &= r_1 + 2r_6 + r_7 \\ s_2 &= r_2 + 2r_7 + r_8 \\ s_3 &= r_3 + 2r_8 + r_9 \\ s_4 &= r_4 + 2r_9 + r_{10} \\ s_5 &= r_5 + 2r_{10} \end{aligned}$$

Note that we can perform a subtraction operation in place of the double operation because of the characteristic of this field and representation as described in Section 3.

5 Timing of Field Operations

In order to show that arithmetic in $\mathbb{F}_{3^n}$ is suitable, in terms of performance and size, for use in cryptosystems, we implemented a number of algorithms in field-programmable hardware. Our algorithms for addition and multiplication were implemented using version 2.1 of the Celoxica [1] Handel-C [2] hardware

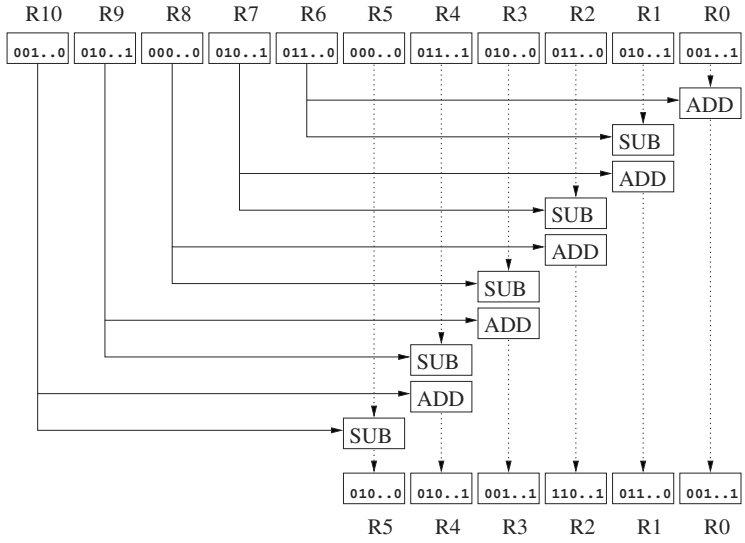

Fig. 4. Reduction Modulo $\theta^6 + \theta + 2$

compilation system and a PCI resident, Xilinx4000XL FPGA based prototyping device [3]. The Handel-C language and compiler tool-chain allowed us to experiment in a familiar high level language, very similar to C, and directly produce hardware implementations from a program in that language. The output of the Handel-C compiler was placed and routed using Xilinx Foundation 3.1i.

All designs communicate input and output data though on-board RAM and use a system clock of 20MHz. We average the results of our timings over 10000 experiments to gain a more representative answer than might otherwise be obtained.

We note that due to our use of a slightly unconventional design process, our results may not be suitable for comparison with, for example, highly optimised VHDL designs. Additionally, we note that we used a somewhat dated version of the Handel-C and Xilinx tool-chains and that more recent versions may offer enhanced optimisation phases which could improve the performance, clock speed and size of our designs. Specifically, we expect to drastically reduce the size of our designs, by using shared arithmetic elements, since the current results are blatantly larger than one might expect. However, we feel that the comparisons offered below are valid in showing both the advantage of our alternative representation and that such arithmetic need not be considered significantly slower than comparable characteristic two alternatives.

In all our experiments, the following notation is used to describe the type of arithmetic being tested

– $\mathbb{F}_{3^{97}} - S$ corresponds to an implementation using the standard software technique of representing each polynomial as an array of 97 integers, where arithmetic is performed using a naive multiplication algorithm.

– $\mathbb{F}_{3^{97}} - B$ refers to our alternative representation using a full bit-serial multiplication method.

The performance for $\mathbb{F}_{2^{241}}$ and $\mathbb{F}_{3^{97}}$ polynomial addition and multiplication, modulo their respective irreducible trinomial, are shown below

Hardware implementation [unoptimised]			
Field	Addition	Multiplication	Slices
$\mathbb{F}_{3^{97}} - S$	$25.29\mu s$	$4393.34\mu s$	2149
$\mathbb{F}_{3^{97}} - B$	$1.20\mu s$	$102.21\mu s$	4136
$\mathbb{F}_{2^{241}}$	$0.80\mu s$	$96.63\mu s$	4920

Notice that addition and multiplication, in our alternative representation of characteristic three, are an order of magnitude faster than the standard $\mathbb{F}_{3^{97}}$ algorithms. Additionally, addition and multiplication are very close to being as fast as arithmetic in $\mathbb{F}_{2^{241}}$.

These addition and multiplication algorithms were implemented with the same basic structure with reduction happening in-place rather than at the end of a multiplication. However, since both the $\mathbb{F}_{3^{97}}$ and $\mathbb{F}_{2^{241}}$ algorithms are bit rather than word oriented, they can easily be accelerated by making size/speed tradeoffs. For example, we can use some extra space to allow reduction to be performed at the end of multiplication and sacrifice further space to add a degree of parallelism to our bit-serial multiplication technique. We also apply additional optimisations which are based on knowledge about how the Handel-C compiler generates hardware for a given input.

By applying these optimisations, we obtain two faster versions of our basic algorithms in both fields

Hardware implementation [optimised]			
Field	Addition	Multiplication	Slices
$\mathbb{F}_{3^{97}}$	$1.15\mu s$	$50.68\mu s$	8733
$\mathbb{F}_{2^{241}}$	$0.70\mu s$	$37.32\mu s$	10139

Since the majority of elliptic curve operations will use these primitives as the basis for more complex operations, the small difference in terms of performance is an important result, it essentially says that characteristic three arithmetic is not necessarily much slower than characteristic two arithmetic.

We can use these optimised addition and multiplication designs as the basis for further algorithms to perform arithmetic in extensions of their respective base fields. We now need to compare arithmetic in $\mathbb{F}_{3^{6 \cdot 97}}$ with arithmetic in $\mathbb{F}_{2^{4 \cdot 241}}$ due to the different values of α in Section 2

Hardware implementation [optimised]			
Field	Addition	Multiplication	Slices
$\mathbb{F}_{3^{6 \cdot 97}}$	$5.90\mu s$	$1843.71\mu s$	10854
$\mathbb{F}_{2^{4 \cdot 241}}$	$3.10\mu s$	$609.04\mu s$	12286

These results show that addition in the two extension fields is roughly equivalent in terms of how long it takes, while using multiplication in $\mathbb{F}_{3^{6\cdot 97}}$ is three times as costly as in $\mathbb{F}_{2^{4\cdot 241}}$. The space required for both implementations is about the same.

Notice that the above implementation used naive arithmetic for performing the extension field multiplication. This was chosen so as to minimise the area of the final hardware solution. Hence, we see that in both cases that if M_b denotes the time needed to perform a base field multiplication and M_e denotes the time needed to perform an extension field multiplication, that

$$M_e \approx n^2 M_b$$

where $n = 6$ in characteristic three and $n = 4$ in characteristic two.

An interesting extension to these results would be to consider the use of Karatsuba multiplication. Although this would lead to a significant increase in area, due to the need to store intermediate results, it could further improve on the arithmetic performance in both fields.

First we deal with the case of even characteristic, where we need to multiply two polynomials of degree three. Using Karatsuba multiplication we can reduce this to three multiplications of polynomials of degree one, plus a little book keeping which we shall ignore. We then multiply the polynomials of degree one, again using Karatsuba, using three base field multiplications. Hence, in characteristic two one expects to obtain

$$M_e \approx 9M_b.$$

In characteristic three we need to multiply two polynomials of degree five over the base field. Using a trivial extension of Karatsuba, which can be found for example in [4] in a similar context, we first apply standard Karatsuba to reduce the problem to the multiplication of three polynomials of degree two. These three products are then computed via performing six base field multiplications each. Hence, in characteristic three one expects to obtain

$$M_e \approx 18M_b.$$

We would therefore expect that a fully optimised version of extension field arithmetic for both characteristics would result in a multiplication algorithm for characteristic three extension fields which is four times slower than the corresponding implementation of characteristic two extension fields. This may not be such a problem in practice as much of the protocols based on the Tate pairing make use of only base field arithmetic, and only the computation of the pairing requires extension field arithmetic. When implementing pairing computations one also attempts to reduce the number of full extension field multiplications that one needs to perform, see [5] and [9] for details.

Finally, to offer further comparison between our techniques, we also implemented them in a software environment. The timings were taken using the same 150MHz Intel PentiumPro equipped FPGA host PC used in the hardware experiments and were compiled using GCC 2.95.1 with all optimisations turned on.

The timings for addition and multiplication in both the base field and extension field are shown below

Software implementation [optimised]		
Field	Addition	Multiplication
$\mathbb{F}_{3^{97}} - S$	$11.89\mu s$	$1013.61\mu s$
$\mathbb{F}_{3^{97}} - B$	$3.98\mu s$	$153.85\mu s$
$\mathbb{F}_{2^{241}}$	$3.31\mu s$	$178.60\mu s$

Software implementation [optimised]		
Field	Addition	Multiplication
$\mathbb{F}_{3^{6 \cdot 97}}$	$8.91\mu s$	$5138.75\mu s$
$\mathbb{F}_{2^{4 \cdot 241}}$	$5.12\mu s$	$3156.86\mu s$

By comparing the results for software and hardware implementation, we can see that in both cases $\mathbb{F}_{3^{97}} - B$ based arithmetic is quicker than a corresponding naive representation. Furthermore, the improvement in the hardware implementation of $\mathbb{F}_{3^{97}} - B$ over $\mathbb{F}_{3^{97}} - S$ is greater than that in software indicating that it is indeed more naturally defined in this medium. Finally, even though our software test environment is far from state of the art, in both cases our hardware implementations significantly out-perform their software equivalents. This is clearly the expected outcome but it is reassuring that even by using an out of date hardware design tool-chain, we were able to produce effective designs using the Handel-C system.

6 Conclusion

We have shown how the use of a novel representation can result in an implementation of characteristic three arithmetic suitable for use in hardware cryptosystems based on the Tate pairing. The use of characteristic three with the Tate pairing is preferred due to the improved bandwidth considerations implied by the security parameters.

Our implementation techniques offer a considerable improvement over the standard techniques based on using a word oriented approach to holding polynomial coefficients. We have also demonstrated that it is possible to implement characteristic three arithmetic which is comparable in performance to a space-equivalent characteristic two alternative. This is a valuable result which allows system designers to benefit from bandwidth reduction without degraded performance.

References

1. Celoxica Technology Overview `http://www.celoxica.com`
2. Celoxica Handel-C Language Overview `http://www.celoxica.com/products/technical_papers/datasheets/DATHNC001_2.pdf`

3. Celoxica Reconfigurable Hardware Development Platform: RC1000 `http://www.celoxica.com/products/technical_papers/datasheets/DATRHD001_2.pdf`
4. D. Bailey and C. Paar. Efficient arithmetic in finite field extensions with application in elliptic curve cryptography. *J. Cryptology*, **14**, 153–176, 2001.
5. P.S.L.M. Barreto, H.Y. Kim and M. Scott. Efficient algorithms for pairing-based cryptosystems. To appear *Advances in Cryptology - CRYPTO 2002*, Springer LNCS 2442, 2002.
6. D. Boneh and M. Franklin. Identity-based encryption from the Weil pairing. In *Advances in Cryptology - CRYPTO 2001*, Springer-Verlag LNCS 2139, 213–229, 2001.
7. D. Boneh, B Lynn and H. Shacham. Short signatures from the Weil pairing. In *Advances in Cryptology - ASIACRYPT 2001*, Springer-Verlag LNCS 2248, 514–532, 2001.
8. S.D. Galbraith. Supersingular curves in cryptography. In *Advances in Cryptology - ASIACRYPT 2001*, Springer-Verlag LNCS 2248, 495–513, 2001.
9. S.D. Galbraith, K. Harrison and D. Soldera. Implementing the Tate pairing. *Algorithmic Number Theory Symposium, ANTS-V*, Springer-Verlag LNCS 2369, 324–337, 2002.
10. K. Harrison, D. Page and N.P. Smart. Software implementation of finite fields in characteristic three. Preprint, 2002.
11. F. Hess. Efficient Identity based Signature Schemes based on Pairings To appear *Selected Areas in Cryptography 2002*.
12. A. Joux. A one round protocol for tripartite Diffie-Hellman. In *Algorithmic Number Theory Symposium, ANTS-IV*, Springer-Verlag LNCS 1838, 385–394, 2000.
13. A.J. Menezes, T. Okamoto and S. Vanstone. Reducing elliptic curve logarithms to logarithms in a finite field. *IEEE Trans. Info. Th.*, **39**, 1639–1646, 1993.
14. V. Miller. Short programs for functions on curves. Unpublished manuscript, 1986.
15. P.L. Montgomery. Modular multiplication without trial division. *Math. Comp.*, **44**, 519–521, 1985.
16. K. Paterson. ID-based Signatures from Pairings on Elliptic Curves. Preprint 2002.
17. R. Sakai, K. Ohgishi and M. Kasahara. Cryptosystems based on pairing. In *SCIS 2000*, 2000.