Pairing Calculation on Supersingular Genus 2 Curves

Colm Ó hÉigeartaigh and Michael Scott

School of Computing, Dublin City University.
Ballymun, Dublin 9, Ireland.
{coheigeartaigh,mike}@computing.dcu.ie

Abstract. In this paper we describe how to efficiently implement pairing calculation on supersingular genus 2 curves over prime fields. We find that, contrary to the results reported in [8], pairing calculation on supersingular genus 2 curves over prime fields is efficient and a viable candidate for practical implementation. We also show how to eliminate divisions in an efficient manner when computing the Tate pairing, and how this algorithm is useful for curves of genus greater than one.

Keywords: Tate pairing, hyperelliptic curves, pairing computation.

1 Introduction

Following a seminal paper by Boneh and Franklin [5] in 2001, there has been an explosion of interest in the exploitation of the properties of bilinear pairings on elliptic curves for cryptographic protocols. Naturally, there has also been much focus on the efficient implementation of pairings. Victor Miller gave the first algorithm [17] for computing a bilinear pairing, specifically the Weil pairing. However in practice the Tate pairing is used, as it is computationally less expensive.

In an important paper, Barreto et. al. [2] gave criteria under which divisions in Miller's algorithm can be eliminated entirely. According to [20], this reduces the calculation time by almost 50%. Other papers which describe important improvements to computing the Tate pairing on elliptic curves are [11] and [3]. Numerous papers describe the actual implementational details, for example see [21].

Although the vast majority of work has been done using elliptic curves, an increasing amount of attention is being focused on computing pairings using hyperelliptic curves of genus 2. Choie and Lee [8] investigate the implementation of the Tate pairing on supersingular genus 2 curves over prime fields, with embedding degree 4. Barreto et. al. [1] describe an efficient implementation of the Tate pairing using the eta pairing construct on supersingular genus 2 curves, over fields of characteristic 2. The significance of this paper is that it not only shows that pairing computation is comparable on genus 2 curves to elliptic curves, but that it can in fact be even faster.

In this paper, we first of all give an improvement to Miller's algorithm for calculating the Tate pairing when working over quadratic extension fields, for arbitrary curves. This algorithm is more efficient than the GHS algorithm [11], but not as efficient as the denominator elimination technique of Barreto et. al [2] in the general case. However, we show that the new algorithm is more efficient than the denominator elimination algorithm for special cases using curves of genus greater than one.

We then report the first efficient implementation of the Tate pairing on genus 2 curves over prime fields. We detail various enhancements to Miller's algorithm that are in the literature, explaining how to apply them to the genus 2 case using prime fields for the first time. Finally, we give more efficient formulae for computing the functions required in Miller's algorithm for the genus 2 case than that reported in [8], the saving being a squaring and a multiplication in each iteration.

This paper is organised as follows. Section 2 gives an overview of the Tate pairing and Miller's Algorithm. Section 3 details an algorithm for computing the Tate pairing, assuming an even embedding degree, without using the two-variable approach of [11]. Section 4 shows how to apply various techniques from the literature for speeding up pairing computation to the specific genus 2 case over prime fields. Section 5 gives experimental results, and we draw our conclusions in section 6.

Section 3 was partly presented in a short paper on the ePrint archive (see [13]), and was presented in the rump session at the ECC 2006 conference.

2 The Tate Pairing

We say that a subgroup of the divisor class group of a hyperelliptic curve C defined over a finite field $\mathbb{F}_p$ has embedding degree k, if the order r of the subgroup divides $p^k - 1$, but does not divide $p^i - 1$ for all $0 < i < k$. The Tate pairing maps the discrete logarithm in the subgroup, to the discrete logarithm in the finite field $\mathbb{F}_{p^k}$, which is the basis of the Frey-Rück attack [10].

Using the notation above, let r be a prime number which is coprime to p. Let $G = J_C(\mathbb{F}_{p^k})$ be the Jacobian Variety of C over $\mathbb{F}_{p^k}$ and let $G[r]$ be the r-torsion group and G/rG the quotient group. Then the Tate pairing is defined as;

$$\langle \cdot, \cdot \rangle_r : G[r] \times G/rG \rightarrow \mathbb{F}_{p^k}^* / (\mathbb{F}_{p^k}^*)^r$$

We follow Galbraith et. al. [11] in defining the first argument over the smaller field $\mathbb{F}_p$ instead of $\mathbb{F}_{p^k}$. This greatly improves computational efficiency, as all the coefficients of the functions in Miller's Algorithm will also be defined over the field $\mathbb{F}_p$. To allow for an efficient implementation of the extension field arithmetic, we also assume that the embedding degree k is even. The Tate pairing as detailed above is only defined up to r-th powers. As a unique value is required for cryptographic purposes, we define the *reduced* pairing;

$$e(D_1, D_2) = \langle D_1, D_2 \rangle_r^{(p^k-1)/r}.$$

The Tate pairing is both well-defined and non-degenerate. However its most important property is *bilinearity*. We define bilinearity for any integer n as, $\langle [n]P, Q \rangle \equiv \langle P, [n]Q \rangle \equiv \langle P, Q \rangle^n$ (modulo r^{th} powers). The Tate pairing can be computed using an algorithm due to Miller [17], as described in Algorithm 1 for an arbitrary hyperelliptic curve. This algorithm is basically the usual ”double and add” algorithm combined with an evaluation of certain intermediate functions (see chapter 9 of [4] for more details).

In Algorithm 1 the divisions are postponed until the end of the loop, to avoid performing a division each loop iteration. To do this we use two variables in the loop, to effectively replace a division with a squaring each loop iteration, which is considerably less expensive to compute. This is an idea given by Galbraith et. al. [11]. However as we shall see in the next section, when the embedding degree is even, this optimisation is unnecessary.

After the main loop, the final exponentiation of $(p^k - 1)/r$ is performed to obtain a unique value over $\mathbb{F}_{p^k}$. It is this unique value which can then be used for cryptographic purposes. It is well known that if arithmetic in $\mathbb{F}_{p^k}$ is implemented using quadratic extensions, an element in this field can be exponentiated to the power of $p^{k/2}$ using a simple conjugation. Conjugation is denoted by $\overline{x} = (a - bi)$ for an element $x = (a + bi) \in \mathbb{F}_{p^k}$.

Taking advantage of this, it is standard to efficiently compute the final exponentiation as $f = \overline{f}/f$ followed by $f = f^{(p^{k/2}+1)/\phi_k(p)}$ and $f = f^{(\phi_k(p)/r)}$, where $\phi_d(x)$ is the d^{th} cyclotomic polynomial. The final $f = f^{(\phi_k(p)/r)}$ exponentiation can be computed efficiently using Lucas Sequences [23, 24].

Algorithm 1 Miller's algorithm to compute the Tate pairing

```
INPUT: $P \in J_C(\mathbb{F}_p), Q \in J_C(\mathbb{F}_{p^k})$ where $P$ has order $r$.
OUTPUT: $e(P, Q)$
$f_c \leftarrow 1, f_d \leftarrow 1$
$T \leftarrow P$
for $i \leftarrow \lfloor \log_2(r) \rfloor - 1$ downto $0$ do
    ▷ Compute $T' = [2]T - div(c/d)$
    $T \leftarrow [2]T$
    $f_c \leftarrow f_c^2 \cdot c(Q)$
    $f_d \leftarrow f_d^2 \cdot d(Q)$
    if $r_i = 1$ then
        ▷ Compute $T' = T + P - div(c/d)$
        $T \leftarrow T + P$
        $f_c \leftarrow f_c \cdot c(Q)$
        $f_d \leftarrow f_d \cdot d(Q)$
    end if
end for
$f \leftarrow f_c/f_d$
$f \leftarrow \overline{f}/f$
$f \leftarrow f^{(p^{k/2}+1)/\phi_k(p)}$
$f \leftarrow f^{(\phi_k(p)/r)}$
return $f$
```

An important improvement on Miller's Algorithm as detailed above was given in [2]. If the x-coordinate of the image point Q is defined over a subfield of $\mathbb{F}_{p^k}$, then the denominator, or the f_d variable in algorithm 1, will also be defined over a subfield of $\mathbb{F}_{p^k}$. This is because the denominator function relies solely on the evaluation at the x-coordinate of Q each iteration. As any value defined over a subfield of $\mathbb{F}_{p^k}$ will be destroyed by the final exponentiation of $(p^{k/2} - 1)$, the f_d variable can be removed from algorithm 2 completely.

Some distortion maps naturally map the x-coordinate of a point to a subfield of $\mathbb{F}_{p^k}$. When this is not the case, a simple transformation of the point as detailed in the next section will have the desired effect. However, as will be seen, this approach is problematic for curves of genus greater than one.

3 Eliminating divisions in Miller's Algorithm

In this section, we show how the denominator elimination technique is problematic with curves of genus greater than one, assuming that a suitable distortion map does not exist. We then present a more efficient algorithm for computing the Tate pairing over quadratic extension fields than is given in algorithm 1, which overcomes the problems associated with denominator elimination in certain contexts.

As seen in the previous section, another approach is required to get the denominator elimination technique to work, when no distortion map exists that maps the x-coordinate of a point from the ground field to a subfield of the field $\mathbb{F}_{p^k}$. Instead we must apply an idea given in a paper by Barreto et. al. [3]. To apply denominator elimination in this case, generate a distorted point Q over $\mathbb{F}_{p^k}$ and get a trace-zero point with: $R = Q - Q^{p^{k/2}}$. The point R will have an x-coordinate defined over a subfield, and so the denominator elimination technique can be used.

However, this technique is problematic when the genus is greater than one, as it increases the weight of the image divisor. For example, if we are evaluating at a degenerate divisor in the genus 2 setting, that consists of a single point on the support, then the above mapping will result in a more general divisor with two points. This will not happen using elliptic curves, as the divisor class group consists solely of divisors with a single point on the support. So in the genus 2 setting, instead of evaluating the functions in Miller's algorithm at a single point, they must be evaluated at two points to use the denominator elimination technique. This drastically reduces the efficiency of denominator elimination.

We now present an alternative way to proceed, by introducing a new variant of Miller's algorithm, assuming that field arithmetic is implemented using quadratic extension fields. In recent papers based on implementing the Weil Pairing [14, 22], the output of the pairing is raised to $(p^{k/2} - 1)$. When this is done, it is possible to replace inversions with conjugations, effectively replacing an expensive operation with one that is free to compute. The following simple lemma is presented as a means of explanation;

Lemma 1. *Let $x = (a + ib) \in \mathbb{F}_{p^k}$, where $a, b \in \mathbb{F}_{p^{k/2}}$. Then the following holds;*

$$(\frac{1}{x})^{p^{k/2}-1} = (\overline{x})^{p^{k/2}-1}$$

Proof. $(\frac{1}{x})^{p^{k/2}-1} = \frac{1}{(a+ib)^{p^{k/2}-1}} = \frac{(a+ib)}{(a-ib)}$. Similarly, $\overline{x}^{p^{k/2}-1} = \overline{(a+ib)}^{p^{k/2}-1} = (a-ib)^{p^{k/2}-1} = \frac{(a+ib)}{(a-ib)}$.

Our observation on how to apply this is a simple one. When computing the Tate pairing, the final exponentiation to obtain a unique r^{th} root of unity includes this value. As seen in the previous section, the final exponentiation can be written as $(p^k - 1)/r = (p^{k/2} - 1)(p^{k/2} + 1)/r$. So as the output of the loop is implicitly raised to the power of $(p^{k/2} - 1)$, there is no need of the GHS strategy of using two variables to eliminate divisions, as a division in the main loop can be replaced by a multiplication and a conjugation. The new algorithm is described in Algorithm 2.

As the variable f_d is eliminated from the pairing calculation, the saving is a squaring over $\mathbb{F}_{p^k}$ each iteration of the loop compared to the GHS approach. This is still not as efficient as performing denominator elimination, which would save a multiplication over this again each iteration. However, when computing the Tate pairing with curves of genus greater than one, and using a distortion map that does not allow denominator elimination directly, algorithm 2 is a slightly more efficient algorithm.

The reason for this is that the denominator elimination algorithm consists of two evaluations at the line function each iteration (or one evaluation of a more complicated form if Mumford representation (see Cantor [6]) is used). Algorithm 2 consists of one evaluation at the line function, and one evaluation at the vertical line function, which requires less computation to evaluate than the line function. Algorithm 2 is also less restrictive than using denominator elimination, as it places no conditions on the form of the image divisor, unlike denominator elimination.

Table 1 illustrates the complexity of the different algorithms in more detail.

Algorithm 2 An improved algorithm for computing the Tate Pairing

```
INPUT: $P \in J_C(\mathbb{F}_p), Q \in J_C(\mathbb{F}_{p^k})$ where $P$ has order $r$.
OUTPUT: $\langle P, Q \rangle_r$
$f \leftarrow 1$
$T \leftarrow P$
for $i \leftarrow \lfloor \log_2(r) \rfloor - 1$ downto $0$ do
    ▷ Compute $T' = [2]T - div(c/d)$
    $T \leftarrow [2]T$
    $f \leftarrow f^2 \cdot c(Q) \cdot \overline{d(Q)}$
    if $r_i = 1$ then
        ▷ Compute $T' = T + P - div(c/d)$
        $T \leftarrow T + P$
        $f \leftarrow f \cdot c(Q) \cdot \overline{d(Q)}$
    end if
end for
$f \leftarrow \overline{f}/f$
$f \leftarrow f^{(p^{k/2}+1)/\phi_k(p)}$
$f \leftarrow f^{(\phi_k(p)/r)}$
return $f$
```

4 Implementing the pairing

In this section, various techniques that allow for an efficient implementation of the Tate pairing using supersingular genus 2 curves over prime fields are described. Timings are given in section 5.

Table 1. Complexity of function calculation in Miller's Algorithm

case	description	complexity
1	Original Approach	1I, 2M, 1S
2	Two-variable Approach	2M, 2S
3	Algorithm 2	2M, 1S
4	Denominator Elimination	1M, 1S

4.1 The Curve

Following [7] and [8], we implement the Tate pairing on the supersingular genus 2 curve;

$$y^2 = x^5 + a,\ a \in \mathbb{F}_p^*,\ p \equiv 2, 3 \mod 5$$

In practice, we take $a = 1$ for convenience. The other supersingular genus 2 curve defined over a prime field with a low embedding degree that was given in [7], is unsuitable for cryptography as it is isogenous to a product of elliptic curves [12].

The order of the Jacobian of this curve is $\#J_C(\mathbb{F}_p) = p^2 + 1$, and hence the embedding degree of the curve is 4. The *distortion map* that maps points on the curve defined over the field $\mathbb{F}_p$ to the larger field $\mathbb{F}_{p^4}$ is given as;

$$\phi(x, y) = (\zeta_5 x, y)$$

where ζ_5 is a primitive 5^{th} root of unity defined over $\mathbb{F}_{p^4}$. Note that ζ_5 maps the x-coordinate to $\mathbb{F}_{p^4}$, and hence does not give denominator elimination directly. Choie and Lee give explicit formulae [8] for calculating the functions required for the genus 2 Tate pairing, which are derived from Tanja Lange's explicit genus 2 formulae [15].

Doubling a divisor is by far the most important part of the group arithmetic for pairings, under the assumption that we are using a prime-order subgroup which has an order of low Hamming Weight. In [8] the given cost of doubling a general divisor (in the overwhelmingly common case) and extracting the functions required for Miller's algorithm is 1 inversion, 23 multiplications and 5 squarings over $\mathbb{F}_p$. However, we save a squaring and a multiplication over this. In table 5 of the appendix, we give the formulae for doubling a general divisor as per [8], with these optimisations built in (the multiplication is saved in step 8). Note our assumption that, as the characteristic of the field is odd, the h polynomial is zero, where the h polynomial comes from the definition of the curve as $y^2 + h(x)y = f(x)$.

We suggest that the formulae in table 5 are optimal, as they have the same computational cost as simply doubling a divisor as given in [15], ie. calculating the functions required for Miller's algorithm is for free.

4.2 Prime-order subgroup

We use the conventions suggested by Lenstra and Verheul [16] and used by Scott [22], to define the levels of security required. We define them as $(160/1024)$, $(192/2048)$ and $(224/4096)$, where the first number in each term is the group size, and the second number is the size of the field kF. As our embedding degree is $k = 4$, we are required to work with finite fields $\mathbb{F}_p$, where $p \sim 256$, 512 and 1024-bits.

When considering what group size to use there are two options, either to use a prime-order subgroup or the full order of the Jacobian. The latter has the advantage that the order of the Jacobian often has a small Hamming weight, and the final exponentiation can be far less expensive. However, if we choose the order of the prime-order subgroup such that it has a low Hamming weight, and if it is far smaller than the order of the Jacobian, then the former method is better.

Rather than using a random prime-order subgroup, we choose a special prime of low Hamming Weight known as a *Solinas* prime [25]. These primes require only two additions in Miller's algorithm. As Duursma and Lee noted [9], the final addition can be skipped assuming that denominator elimination is applied. See section 5 for further details on the parameters used.

4.3 Finite Field Representation

The best way to represent elements of the field $\mathbb{F}_{p^4}$ is to represent them as a quadratic extension of $\mathbb{F}_{p^2}$, which is in turn a quadratic extension of $\mathbb{F}_p$. If the prime p is congruent to $1 \mod 4$, then the irreducible polynomial $x^2 + 2$ can be used to represent the quadratic extension field $\mathbb{F}_{p^2}$.

So, assuming that $i = -2$ is a quadratic non-residue, we represent elements of the field $\mathbb{F}_{p^2}$ as $(a + b\sqrt{i})$, where $a, b \in \mathbb{F}_p$, and we represent elements of the field $\mathbb{F}_{p^4}$ as $(c + d\sqrt[4]{i})$, where $c, d \in \mathbb{F}_{p^2}$. As our curve is defined over a prime $p \equiv 2, 3 \mod 5$, we will use a prime $p \equiv 5 \mod 8$ to take advantage of our finite field representation. An advantage of using a prime of this form is the resulting simple formula for modular square roots, as required for generating points on the curve.

4.4 Using Degenerate Divisors

Duursma and Lee [9] introduced the notion of working with a degenerate divisor for pairing applications with curves of genus greater than 1. In the genus 2 context, we will define a degenerate divisor as a divisor with only one point in its support, rather than the more general two. There is no advantage to be gained in using a degenerate divisor as the first argument to Miller's algorithm, as with the first doubling the divisor will turn into a more general divisor with two points on the support, unless one takes advantage of an automorphism that keeps the divisor in its special shape (eg. see [1]).

However, as we are evaluating the second divisor at a function, we achieve a speedup by evaluating at a degenerate divisor (ie. a single point). Evaluating at a more general divisor requires evaluation at two points, or else using the divisor's Mumford representation.

4.5 Evaluating functions

Each iteration of the loop requires the evaluation of the function $f_c = y_1 - ((x_1\zeta_5)^3 s_1 + (x_1\zeta_5)^2 l_2 + (x_1\zeta_5)l_1 + l_0)$, where s_1, l_0, l_1, l_2 are from Cantor's algorithm, ζ_5 is a primitive 5^{th} root of unity defined over $\mathbb{F}_{p^4}$ and x_1 is the x-coordinate of the point at which we are evaluating. As $(x_1\zeta_5)^3$, $(x_1\zeta_5)^2$ and $(x_1\zeta_5)$ can be precomputed, this leaves 12 multiplications over $\mathbb{F}_p$ to be computed each time the function is evaluated. However, a multiplication may be saved by examining relations between various powerings of the 5^{th} root of unity.

If ζ_n is a primitive n^{th} root of unity in a field K, then its conjugates over the prime subfield K_0 of K are also primitive n^{th} roots of unity [18]. Also, ζ_n^a is a primitive n^{th} root of unity if and only if a and n are coprime. In our case, the third power of a 5^{th} primitive root of unity over $\mathbb{F}_{p^4}$ is related to the second power by conjugation: $\zeta_5^3 = \overline{\zeta_5^2}$

So instead of evaluating an equation of the form $a + b\zeta_5 + c\zeta_5^2 + d\zeta_5^3$, let the real component of ζ_5^2 be r and the imaginary component i (both of which are defined over $\mathbb{F}_{p^2}$), and we can compute $a + b\zeta_5 + ((c+d) * r, (c-d) * i)$. Computing $b = x_1 l_1$, $c = x_1^2 l_2$ and $d = x_1^3 s_1$ takes 3 multiplications in total (assuming the precomputation of the powers of x_1), thus the total multiplication count in evaluating the function is 11 multiplications, a saving of one multiplication.

4.6 Using Denominator Elimination

As detailed in section 3, it is more efficient to use Algorithm 2 than denominator elimination, assuming a distortion map that does not give denominator elimination directly, and that the image divisor is a degenerate divisor. However, it is possible to reduce the performance gap by using customized multiplication routines, as detailed in this section.

Given a point $Q = (x, y) \in \mathbb{F}_{p^4}$, the transformation $R = Q - Q^{p^2}$ gives a divisor R suitable for use with denominator elimination. Writing this as $R = (x, y) + (\overline{x}, -y)$ avoids using Cantor's algorithm over $\mathbb{F}_{p^4}$ and keeps the two points on the support of the divisor separate. A benefit of this approach is that the function calculated in the main doubling loop, $f_c = y_1 - ((x_1\zeta_5)^3 s_1 + (x_1\zeta_5)^2 l_2 + (x_1\zeta_5)l_1 - l_0)$, can be reused for the calculation of the required function for the second point.

Let the function $f_c = ((a, b), (c, d))$ for the first point (x, y). Then, for the second point $(\overline{x}, -y)$, the function $f_c = ((a - y - y, b), \overline{(c, d)})$. So the calculation of the second function is effectively for free, as it simply involves two subtractions and a conjugation using the function generated by the first point. However, we still have to multiply the two functions together.

In each iteration of the loop, the function f_c is evaluated twice, ie. at the two points. As seen above, the two functions are closely related. It is possible to exploit these relations to speed-up the calculation. So instead of calculating each function separately and multiplying them separately by the overall accumulating variable, we multiply the functions f_{c_1} and f_{c_2} together first, before multiplying the result with the accumulating function.

Normally, multiplying two general elements in $\mathbb{F}_{p^4}$ can be done with only 9 multiplications over $\mathbb{F}_p$, using the Karatsuba technique. For the functions $f_{c_1} = (a, ib)$ and $f_{c_2} = (c, -ib)$, where $a, b, c \in \mathbb{F}_{p^2}$, the $f_{c_1} f_{c_2}$ multiplication is unrolled as;

$$(a, ib)(c, -ib) = ac + 2b^2 + b(c - a)$$

Note that $(c - a) \in \mathbb{F}_p$, rather than $\mathbb{F}_{p^2}$. We can also take advantage of the form of the ac multiplication, where;

$$ac = (e, if)(g, if) = eg - 2f^2 + if(e + g)$$

So the total cost is $2M + S$ for the ac multiplication, plus $2M + 2M$ for the overall multiplication, which results in $6M + S$ instead of the general $9M$. When this technique is used, the performance of denominator elimination and algorithm 2 is roughly the same, for the genus 2 case under consideration. However, we suggest that algorithm 2 is a more natural algorithm to use in practice, as it is simpler to implement than the type of customised multiplication routines given in this section.

4.7 Lucas Exponentiation

As detailed in algorithm 1, the final exponentiation is split into two parts. The first part can be computed with a conjugation and division, and then Lucas exponentiation, as detailed in the paper by Scott and Barreto [23], is used for the $(p^2 + 1)/r$ exponentiation.

4.8 Coding Issues

We use MIRACL [19] to provide the cryptographic primitives needed. In particular, we make use of special assembly-language routines that MIRACL provides, which can be used when working with moduli with a fixed number of bits. All of the implementation was written in C/C++ and timed on a Pentium IV, 2.8 Ghz.

5 Experimental Results

In this section, we give experimental results for computing the Tate pairing using the techniques detailed in this paper for the supersingular genus 2 curve defined over $\mathbb{F}_p$, as defined in section 4.1. We will use the three different levels of security defined earlier for testing, namely $(160/1024)$, $(192/2048)$ and $(224/4096)$.

The only condition on our prime-subgroup order r is that it be congruent to $1 \mod 4$. r, and not r^2, must divide the order of the Jacobian, $p^2 + 1$. The prime p must be congruent to $5 \mod 8$, and also congruent to 2 or $3 \mod 5$, for reasons stated earlier. The parameters used are detailed in appendix A.

Table 2 details the timings for the implementation of the Tate pairing for the $(160/1024)$ security level, table 3 is for the $(192/2048)$ case, and table 4 is for the $(224/4096)$ case. There are four cases in each table. The first is evaluating using a single point. The second case is evaluating at the more general two points, which is the case when the divisor is the sum of two rational points. The third case uses Mumford representation, instead of keeping the points separate. This case has the advantage that it also handles the case when the points on the divisor are defined over a larger field. All these cases use algorithm 2. The fourth cases are timings that are reported in [22] using elliptic cases, with the equivalent level of security.

In the Choie and Lee paper [8], they give implementations of the pairing that range between 500 and 600 ms on a Pentium 4 2 Ghz for a $(160/1024)$ bit security level. However, as seen in table 2, our timings far outperform this. These timings indicate that genus 2 pairings over prime fields are valid candidates for practical implementations. However, as can be seen from the tables, the elliptic cases are roughly twice as fast as the genus 2 timings. This is roughly what we would expect, due to the more complicated group law in the genus 2 case.

Table 2. Running times - $(256/1024)$ security level

case	description	pairing time (ms)
1	evaluating at degenerate divisor	16
2	evaluating at general divisor	20.7
3	evaluating using Mumford rep	20.45
4	elliptic curve timing [22]	8.9

Table 3. Running times - $(192/2048)$ security level

case	description	pairing time (ms)
1	evaluating at degenerate divisor	49
2	evaluating at general divisor	62
3	evaluating using Mumford rep	61
4	elliptic curve timing [22]	20.5

6 Conclusion

We have introduced a new variant of Miller's algorithm over quadratic extension fields by showing how divisions can always be eliminated. This algorithm is not as fast as

Table 4. Running times - $(224/4096)$ security level

case	description	pairing time (ms)
1	evaluating at degenerate divisor	183
2	evaluating at general divisor	232
3	evaluating using Mumford rep	229
4	evaluating at degenerate divisor (denom elim)	175

using denominator elimination in the general case, but can be faster when working with curves of genus greater than one and distortion maps of a certain form.

We have implemented the Tate pairing on supersingular genus 2 curves over prime fields, detailing various optimisations, and showing how genus 2 curves over prime fields are valid candidates for pairing implementation.

7 Acknowledgements

We would like to thank Steven Galbraith and Caroline Sheedy for comments on this paper, and Noel McCullagh for pointing out an error in a previous draft.

References

1. P. S. L. M. Barreto, S. Galbraith, C. Ó hÉigeartaigh, and M. Scott. Pairing computation on supersingular abelian varieties. Cryptology ePrint Archive, Report 2004/375, 2004. Available from `http://eprint.iacr.org/2004/375`.
2. P. S. L. M. Barreto, H. Y. Kim, B. Lynn, and M. Scott. Efficient algorithms for pairing-based cryptosystems. In *Advances in Cryptology – Crypto'2002*, volume 2442 of *Lecture Notes in Computer Science*, pages 354–368. Springer-Verlag, 2002.
3. P. S. L. M. Barreto, B. Lynn, and M. Scott. On the selection of pairing-friendly groups. In *Selected Areas in Cryptography – SAC'2003*, volume 3006 of *Lecture Notes in Computer Science*, pages 17–25. Springer-Verlag, 2003.
4. I. F. Blake, G. Seroussi, and N. P. Smart. *Advances in elliptic curve cryptography*. Cambridge, 2005.
5. D. Boneh and M. Franklin. Identity-based encryption from the Weil pairing. *SIAM Journal of Computing*, 32(3):586–615, 2003.
6. D. G. Cantor. Computing in the jacobian of a hyperelliptic curve. *Math. Comp.*, 48(177):95–101, 1987.
7. Y. Choie, E. Jeong, and E. Lee. Supersingular hyperelliptic curves of genus 2 over finite fields. Cryptology ePrint Archive, Report 2002/032, 2002. `http://eprint.iacr.org/2002/032`.
8. Y. Choie and E. Lee. Implementation of tate pairing on hyperelliptic curves of genus 2. In *Information Security and Cryptology - ICISC 2003*, volume 2971 of *Lecture Notes in Computer Science*, pages 97–111. Springer-Verlag, 2004.
9. I. Duursma and H.-S. Lee. Tate pairing implementation for hyperelliptic curves $y^2 = x^p - x + d$. In *Advances in Cryptology – Asiacrypt'2003*, volume 2894 of *Lecture Notes in Computer Science*, pages 111–123. Springer-Verlag, 2003.

10. G. Frey and H.-G. Rück. A remark concerning m-divisibility and the discrete logarithm problem in the divisor class group of curves. *Math. Comp.*, 52:865–874, 1994.
11. S. Galbraith, K. Harrison, and D. Soldera. Implementing the Tate pairing. In *Algorithmic Number Theory – ANTS V*, volume 2369 of *Lecture Notes in Computer Science*, pages 324–337. Springer-Verlag, 2002.
12. S. D. Galbraith. Personal communication, 2005.
13. C. Ó hÉigeartaigh. Speeding up pairing computation, 2005. `http://eprint.iacr.org/2005/293`.
14. Neal Koblitz and Alfred Menezes. Pairing-based cryptography at high security levels. Cryptology ePrint Archive, Report 2005/076, 2005. `http://eprint.iacr.org/`.
15. T. Lange. Formulae for arithmetic on genus 2 hyperelliptic curves. In *Applicable Algebra in Engineering, Communication and Computing*, Online publication. Springer-Verlag, 2004. `http://www.springerlink.com/openurl.asp?genre=article&id=doi:10.1007/s0%0200-004-0154-8`.
16. A. K. Lenstra and E. R. Verheul. Selecting cryptographic key sizes. *Journal of Cryptology*, 14(4):255–293, 2001.
17. V. S. Miller. Short programs for functions on curves. Unpublished manuscript, 1986. `http://crypto.stanford.edu/miller/miller.pdf`.
18. P. Ribenboim. *Classical Theory of Algebraic Numbers*. Springer-Verlag, 2001.
19. M. Scott. Miracl (multiprecision integer and rational arithmetic c/c++ library). Available from `http://indigo.ie/~mscott/`.
20. M. Scott. Faster identity based encryption. *Electronics Letters*, 40(14):861, 2004.
21. M. Scott. Computing the tate pairing. In *Topics in Cryptology - CT-RSA 2005*, volume 3376 of *Lecture Notes in Computer Science*, pages 293–304. Springer-Verlag, 2005.
22. M. Scott. Scaling security in pairing-based protocols. Cryptology ePrint Archive, Report 2005/139, 2005. `http://eprint.iacr.org/`.
23. M. Scott and P. Barreto. Compressed pairings. In *Advances in Cryptology – Crypto' 2004*, volume 3152 of *Lecture Notes in Computer Science*, pages 140–156. Springer-Verlag, 2004. Also available from `http://eprint.iacr.org/2004/032/`.
24. P. Smith and C. Skinner. A public-key cryptosystesm and a digital signature system based on the lucas function analogue to discrete logarithms. In *Advances in Cryptology – Asiacrypt'1994*, Lecture Notes in Computer Science, pages 357–364. Springer-Verlag, 1995.
25. J. Solinas. Generalized mersenne numbers. Technical Report CORR 99-39, University of Waterloo, 1999. Available from `http://www.cacr.math.uwaterloo.ca/techreports/1999/corr99-39.pdf`.

A Curve Parameters

Here we give the subgroup and prime field parameters that were used for the different security levels;

160/1024 security level:

$r = 2^{159} + 2^{17} + 1$
$p = 633245314511811482002751717312031257188556244933390653108784593318867170658 93$

192/2048 security level:

$r = 2^{191} + 2^2 + 1$
$p = 8928465122808378842689950368414551548287912471534562510973748060201641117468953363599067244027908076232256944699958875614648564192943960634648749730387013$

224/4096 security level:

$r = 2^{223} + 2^{13} + 1$
$p = 155722884131515840187323558851704700783145211009055018661797977213059964066609221691524801350598779752866480421078369507449219791754684643397404851273095293761493705843127836052457915167872334351960770506641541305942224943595487772602516676106413200532581353024750990143717859982402535061826066311255496083453$

B Doubling Formulae

Table 5. Formulae for doubling when deg $u_1 = 2$, gcd$(u_1, 2v_1) = 1)$

Input	$D_1 = [u_1, v_1]$ where $u_1 = x^2 + u_{11}x + u_{10}, v_1 = v_{11}x + v_{10}, f = x^5 + a$	
Output	$D_3 = [u_3, v_3], l(x)$ such that $D_3 + div((y-l)/u_3) = 2D_1$.	
Step	Expression	Cost
1	Compute $\tilde{v_1} \equiv (2v_1)(\mod u_1) = \tilde{v_{11}}x + \tilde{v_{10}}$ $\tilde{v_{11}} = 2v_{11}, \tilde{v_{10}} = 2v_{10}$	
2	Compute $r = res(u_1, \tilde{v_1})$ $w_0 = \tilde{v_{11}}^2, w_1 = u_{11}^2, w_2 = 4w_0, w_3 = u_{11}\tilde{v_{11}},$ $r = u_{10}w_2 + \tilde{v_{10}}(\tilde{v_{10}} - w_3)$	2S + 3M
3	Compute almost inverse of $inv' = r(2v_1)^{-1}(\mod u_1)$ $inv'_1 = -\tilde{v_{11}}, inv'_0 = \tilde{v_{10}} - w_3$	
4	Compute $k' = \frac{F - v_1^2}{u_1}(\mod u_1) = k'_1x + k'_0$ $w_3 = w_1, w_4 = 2u_{10}, k'_1 = 2w_1 + w_3 - w_4$ $k'_0 = u_{11}(2w_4 - w_3) - w_0$	1M
5	Compute $s' = k'inv'(\mod u_1)$ $w_0 = k'_0inv'_0, w_1 = k'_1inv'_1$ $s'_1 = \tilde{v_{10}}k'_1 - \tilde{v_{11}}k'_0, s'_0 = w_0 - u_{10}w_1$ If $s'_1 = 0$ then goto step $6'$.	5M
6	Compute $s = s_1x + s_0$ and s_1^{-1} $w_1 = (rs'_1)^{-1}, w_2 = s'_1w_1, w_3 = r^2w_1,$ $s_1 = s'_1w_2, s_0 = s'_0w_2$	1I, 1S, 5M
7	Compute $l(x) = su_1 + v_1 = s_1x^3 + l_2x^2 + l_1x + l_0$ $l_2 = s_1u_{11} + s_0, l_0 = s_0u_{10} + v_{10}$ $l_1 = (s_1 + s_0)(u_{11} + u_{10}) - s_1u_{11} - s_0u_{10} + v_{11}$	3M
8	Compute $u' = monic(\frac{F - l^2}{u_1^2}) = x^2 + u_{31}x + u_{30}$ $u_{30} = w_3(2v_{11} + w_3(2u_{11} + s_0^2)$ $u_{31} = 2s_0 - w_3$	1S + 2M
9	Compute $v_3 = -l(\mod u_3) = v_{31}x + v_{30}$ $w_1 = u_{31}, u_{31} = w_3u_{31}, w_3 = l_2 - w_1, w_3 = u_{30}w_2$ $v_{31} = (u_{31} + u_{30})(w_2 + s_1) - w_3 - w_1 - l_1, v_{30} = w_3 - l_0$	3M
		1I, 4S, 22M
6’	Compute $l(x) = s_0u_1 + v_1$ $inv = 1/r, s_0 = s'_0inv, l_1 = s_0u_{11} + v_{11}, l_0 = s_0u_{10} + v_{10}$	1I + 3M
7’	Compute $u_3 = monic(\frac{F - l^2}{u_1^2}) = x + u_{30}$ $u_{30} = -2u_{11} - s_0^2$	1S
8’	Compute $v_3 = -l(\mod u_3) = v_{30}$ $v_{30} = u_{30}(l_1 - u_{30}s_0) - l_0$	2M
		1I, 3S, 14M