Implementing Cryptographic Pairings

Michael Scott

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

Abstract. In this informal working draft we attempt to review the state-of-the-art in pairing implementation. Starting with a basic Miller algorithm for the Tate pairing we show how to successively apply a series of optimizations and tricks to improve performance. We will concentrate on the case of non-supersingular prime characteristic elliptic curves, although many of the optimizations equally apply to the cases of supersingular elliptic and hyperelliptic curves. We also discuss optimal implementation of extension field arithmetic.
Keywords: Tate pairing implementation, pairing-based cryptosystems.

1 Introduction

The Tate pairing, denoted $e(P,Q)$, where the points P and Q are linearly independent points on an elliptic curve $E(\mathbb{F}_{q^k})$ evaluates as an element of an extension field $\mathbb{F}_{q^k}$. If P is of prime order r, then the pairing evaluates as an element of order r. This feature can be used to transfer the discrete logarithm problem from the elliptic curve setting, to the easier finite field setting, an ability exploited by Menezes, Okamoto and Vanstone [30], and by Frey, Müller and Rück [17], to solve the elliptic curve discrete logarithm problem in certain instances. The embedding degree k is defined as the *smallest* value of k such that $r|q^k-1$. For a random non-supersingular curve this is unlikely to be the case for a small value of k. However for supersingular curves and specially constructed pairing friendly non-supersingular curves, k will be small. Here we focus on the case of non-supersingular elliptic curves of prime characteristic, so from here on $q=p$. Some actual implementations of pairings are available [39], [29].

The most useful property of the pairing is its *bilinearity*

$$e(aP,bQ) = e(P,Q)^{ab}$$

How to calculate the pairing? In the beginning there was Miller's algorithm. Here we start with a generic implementation in the context of calculating the Tate pairing (which appears to be superior to the Weil pairing for all cases of interest). As is well known in this case a "final exponentiation" is required to obtain a unique result. See algorithm 1.

Observe that the point P is being implicitly multiplied by its group order r using a classic double-and-add line-and-tangent algorithm, until it finally ends

Algorithm 1 Computation of $e(P,Q)$ using basic Miller's algorithm

INPUT: $P \in E(\mathbb{F}_{p^k}), Q \in E(\mathbb{F}_{p^k})$, where P has order r
OUTPUT: $e(P,Q)$

$T \leftarrow P$, $f \leftarrow 1$
for $i \leftarrow \lfloor \lg(r) \rfloor - 1$ **downto** 0 **do**
    $f = f^2.l_{T,T}(Q)/v_{2T}(Q)$
    $T = 2T$
    if $r_i = 1$ **then**
        $f = f.l_{T,P}(Q)/v_{T+P}(Q)$
        $T = T + P$
    end if
end for
$f \leftarrow f^{(p^k-1)/r}$
return f

up at the point-at-infinity. The values of the line and vertical functions $l_{A,B}(Q)$ and $v_{A+B}(Q)$ respectively, are distances calculated between the fixed point Q and the lines that arise when adding B to A on the elliptic curve in the standard way. If the point A has coordinates (x_j, y_j), the point $A + B$ has coordinates (x_{j+1}, y_{j+1}), the point Q has the coordinates (x_Q, y_Q), and the line through A and B has a slope of λ_j, then explicitly

$$l_{A,B}(Q) = (y_Q - y_j) - \lambda_j(x_Q - x_j)$$

$$v_{A+B}(Q) = (x_Q - x_{j+1})$$

Here we assume the use of affine coordinates, although in most cases projective coordinates are to be preferred, with minor modifications to these formulae.

The eagle-eyed may have spotted that the pairing seems to require another parameter, the order r of P. In fact knowledge of r is not strictly required, as the pairing can be calculated without it. Simply remove the final exponentiation, and replace the final index of the Miller loop with $\lg(p^k-1)$ instead of $\lg(r)$. In fact the work load can be shifted between the Miller loop and the final exponentiation by replacing the index with $\lg(mr)$, and the final exponentiation by $(p^k-1)/(mr)$, for any m for which $(p^k-1)/)(mr)$ is a whole number. However it is usual that r is known, in which case concentrating as much as the work as possible into the final exponentiation will be faster.

In fact the algorithm as described may fail catastrophically for random choices of P and Q, as it is quite possible that the tangent or addition line may pass directly through Q, or the vertical function may evaluate as zero. In the sequel this possibility will be side-stepped by choosing P and Q from particular disjoint groups.

Our first couple of optimizations are simple and rather obvious [19], [3]. First we will (to the extent that we can) choose r to have a low Hamming weight (or indeed mr if that has a lower hamming weight for small m). This will clearly result in a faster algorithm. In some cases (e.g. for the MNT curves [32]) the

choice of a low Hamming weight order may not be practical, in which case the optimal strategy might be to represent r in a NAF format, and use a standard windowed NAF double-add-and-subtract algorithm as used for standard elliptic curves. Alternatively the method of Eisentrager et al. [15] might be useful. This will lead to a more complex algorithm that would muddy the waters for us in what we are trying to do here. Therefore we will proceed on the assumption that a low hamming weight r is possible, and even if it is not the further optimizations that we will describe are still applicable.

Secondly we will choose P to be a point on the curve taken over the base field $E(\mathbb{F}_p)$ (Solinas's *Miller light*), with obvious performance advantages. A useful side effect is that the algorithm failure described above cannot now happen (for $k > 1$) if Q is a general point over the full extension field.

We will however resist the adoption of an optimization once suggested by Koblitz and Menezes that the modulus p should also be chosen as being of low Hamming weight as this can introduce plausible security concerns [34].

2 A restriction

Before proceeding we will introduce a useful restriction. From here we assume that k is even, and that $k = 2d$. As we will see this brings many advantages.

The final exponentiation can now be written as $f^{(p^d-1)(p^d+1)/r}$. (And we know that r must divide the $p^d + 1$ part, as otherwise the definition of k above would be violated). As we will see this simple observation facilitates many of our optimizations.

3 Extension field arithmetic

Before proceeding to optimize the pairing algorithm itself, we pause to consider the implementation of extension field arithmetic. This has arisen before in the context of cryptography, notably in the XTR scheme [27], and for elliptic curves implemented over Optimal Extension Fields [1]. However in the former case only quadratic extensions are considered, and in the latter the context is rather specialised by the preference for a small word-sized modulus p of simple form. A much wider range of possible extension fields need to be considered in the context of pairings.

First we need a suitable *irreducible polynomial*, which has a degree the same as our chosen extension field, but which has no factors over the base field.

As an example consider the case where $k = 2$. In this case if $p = 3 \bmod 4$, then we might choose as our irreducible polynomial $x^2 + 1$, because -1 is a quadratic non-residue modulo p, and so the polynomial does not factor over the base field. Now elements in $\mathbb{F}_{p^2}$ can be represented as polynomials like $a+xb$ with $a, b \in \mathbb{F}_p$. Such elements can be multiplied as normal, and then reduced modulo $x^2 + 1$. This means that the occurrence of x^2 in the product may be replaced by -1. Note that x can be considered as the imaginary root of the irreducible

polynomial, in which case elements of the extension field can be represented as $a+ib$, where i is the imaginary square root of -1, and the analogy with complex numbers is exact (and quite comforting!).

The addition and subtraction algorithms are quite obvious and cheap. Division (which turns out not to be time-critical) can be implemented only ever requiring one base field inversion [28]. It is however worth taking a little care over multiplication and squaring. We will assume that a base field squaring (called a modsqr) costs about 0.9 of a base field multiplication (a modmul), a figure typical of practical finite field implementations.

3.1 Multiplication and Squaring

The naive way to do multiplication is

$$(a + ib)(c + id) = ac - bd + i(bc + ad)$$

at a cost of 4 base field modmuls. But of course Karatsuba will be our friend in this setting, and so

$$(a + ib)(c + id) = ac - bd + i[(a + b)(c + d) - ac - bd)]$$

which only requires three modmuls. However a modmul can be divided into separate multiplication and reduction steps. For a prime characteristic field, using Montgomery's method [33] (and recalling that we have eschewed the possibility of a special form for the modulus p on the grounds that it may weaken our security), reduction will cost about the same as a multiplication. In this case the method of "lazy reduction" [28] can be applied, whereby we calculate the full values of the real and imaginary parts of our product before reducing them separately modulo p. In this way only two reductions are required in the above formula, and the overall cost falls to about that of 2.5 base field modmuls.

In the case of squaring we could also apply Karatsuba for a cost of two modsqrs and one modmul. However a better idea would be to use this identity well known to implementors of complex arithmetic

$$(a + ib)(a + ib) = (a + b)(a - b) + i.2ab$$

which requires just two modmuls. Note that for an alternative irreducible polynomial the same basic costs apply, perhaps with more additions and subtractions.

Turning next to the case of the cubic extension, multiplication can be carried out using Karatsuba and 6 modmuls. Using the method of Toom-Cook this can be reduced to 5 modmuls, with some tricky multiplications and divisions by small constants (and divisions are particularly difficult). The idea of lazy reduction can again be used here to advantage. For squaring the best algorithm was only very recently discovered by Chung and Hasan [10]. Assume an irreducible polynomial of the simple form $x^3 + n$, and consider the calculation of $(a + bx + cx^2)^2$. First

precalculate $A = a^2$, $B = 2bc$, $C = c^2$, $D = (a - b + c)^2$ and $E = (a + b + c)^2$, then

$$(a+bx+cx^2)^2 = (A-Bn)+((E-D)/2-B-nC)x+(E-A-C-(E-D)/2)x^2$$

which requires only 4 modsqrs and 1 modmul (and a division by 2 and a lot of adds/subtracts).

What about those annoying divisions by constants that occur for Toom-Cook and for the Chung-Hasan formula above? Well recall that the final exponentiation in the Tate pairing involves exponentiation by a power of $p^d - 1$. This always contains as a factor $p-1$. Recall next Fermat's little theorem ($y^{p-1} \bmod p = 1$). This implies that we are free to include a constant factor into any extension field multiplication or squaring, on the basis that the contribution of this constant will be "wiped-out" in the final exponentiation. Therefore divisions by small constants can always be replaced by multiplications by small constants (in our context), which in turn can be replaced by additions.

3.2 Inversion

For completeness we include formulae for inversion in the quadratic and cubic extension fields. These are quite easy to derive [28].

For the quadratic case, and assuming an irreducible polynomial of $x^2 + n$

$$1/(a + bx) = (a - ib)/(a^2 + nb^2)$$

For the cubic case, and assuming an irreducible polynomial of $x^3 + n$. First precalculate $A = a^2 + nbc$, $B = -nc^2 - ab$, and $C = b^2 - ac$. Then $F = -nbC + aA - ncB$, and

$$1/(a + bx + cx^2) = (A + Bx + Cx^2)/F$$

3.3 Square Roots

In many early protocols there was a requirement to hash values to curves points, e.g. Boneh and Franklin IBE [8]. This practise now seems to be deprecated in favour of hashing to point multipliers, which is much easier to do. However hashing to a curve point is perhaps not quite as complex as is thought. The issue of a "large cofactor" can be dealt with by exploiting the feature of the Tate pairing that it does not require the parameter Q to be of order r for bilinearity to hold (it is sufficient that Q be a representative of a coset – in effect any point on the curve) [36].

If hashing to a curve point there may be a requirement to hash to a point on the curve over an extension field. This usually requires the ability to extract square roots over the extension, although for some curves cube rooting may be more useful [8], [6]. In the quadratic extension case (again borrowing from standard methods for complex arithmetic [18]) we have the nice identity

$$\sqrt{a+xb} = \pm(\sqrt{(a \pm \sqrt{a^2+nb^2})/2} + xb/(2\sqrt{(a \pm \sqrt{a^2+nb^2})/2}))$$

which is a lot simpler than the method suggested by Wang et al. [43]. Only two square roots are required over the base field. Whether or not $a + xb$ is a quadratic residue is determined solely by whether or not $a^2 + nb^2$ is a quadratic residue (the proof left as an exercise to the reader), so the quadratic residuosity test is in this case particularly efficient.

For the cubic extension case no simple closed formula for the square root seems to be possible. Over the field $\mathbb{F}_{p^3}$ for $p = 3 \bmod 4$ we can offer the formula

$$\sqrt{u} = \pm(u^{p^2}(u^p)^3 u)^{(p-3)/4} u(u^p)^2$$

assuming of course that u is a quadratic residue. A similar formula requiring primarily the work of one simple exponentiation can also be derived for the $p = 5 \bmod 8$ case.

$$v = ((2u)^{p^2}((2u)^p)^5 2u)^{(p-5)/8}((2u)^p)^3$$
$$\sqrt{u} = \pm(uv(2uv^2 - 1))$$

For the remaining $p = 1 \bmod 8$ case a variant of the standard Tonelli-Shanks algorithm can be used [31]. The Frobenius (see below) is very helpful in deriving these formulae.

For a discussion of cube roots in quadratic extensions, see Appendix B of [6].

3.4 The Frobenius action

Of particular importance in extension field arithmetic is the "Frobenius action". Raising an extension field element to the power of the modulus is always very cheap. Going back for a moment to our simple example above, we have the relationship

$$(a + ib)^p = (a^p + i^p.b^p) = (a - ib) \bmod p$$

This is quite easy to prove, the "other" terms is the expansion of $(a+ib)^p$ are all zero modulo p, and recall that $i^{(p-1)/2} = -1$ as i is a quadratic non-residue.

3.5 A Tower of extensions

For higher extension fields, a "tower of extensions" can be used, and these methods for multiplication, squaring, inversion and square rooting can be used recursively. For example consider the sextic extension with the irreducible polynomial $x^6 + n$. Using the Chung-Hasan idea and building a cubic extension on top of a quadratic extension, a squaring in this field will only require 11 base field modmuls.

For most cases of interest we can further restrict $k = 2^i 3^j$ for $i \geq 1$ and $j \geq 0$ [26]. In these cases we only need to consider quadratic or cubic extensions of the field below.

However more work is needed to determine which is the best method to use for higher extension degrees. Which is better Karatsuba or Toom-Cook? How best to organise the towering? It seems that for MNT $k = 6$ curves that a quadratic layered on top of a cubic works best. For $k = 12$ BN curves (see below) a quadratic over a cubic over a quadratic works well in practise. Of course it is possible to switch from one representation to another, but in the interests of keeping the code elegant and minimizing code-size it would be nice to avoid it if possible. Some testing of the various alternatives has already been carried out and reported in [12].

4 Another restriction

Koblitz and Menezes [26] have proposed the use of "pairing-friendly" fields, where $p = 1 \bmod 12$ and the irreducible polynomial is of the form $x^k + \beta$, where $\beta \in \mathbb{F}_p$. In our view the $p = 1 \bmod 12$ condition is quite restrictive, and the value of β for a particular p can be quite large, leading to more additions at the bottom of the tower, and hence less efficient implementations.

Instead we recommend a type of construction like that proposed by Barreto and Naehrig [6] for use with the BN family of pairing friendly curves (see below). We will continue to insist that the irreducible polynomials at each level in the tower are binomial, and furthermore that the imaginary roots of these polynomials should contain the same roots as those used at the lower level. For example for $k = 12$ we could have an irreducible polynomial of the form $X^6 + (\alpha + \sqrt{-\beta})$ as a sextic extension built on top of a quadratic extension which has as an irreducible polynomial $x^2 + \beta$. This particular construction is ideal for the BN curves, allows the use of $p = 3 \bmod 4$ curves, and often permits the use of very small $|\alpha|, |\beta| \leq 2$.

5 Types of curves

Pairing-friendly curves have two parameters of relevance to us here. The first ρ is the rounded-to-nearest-simple-fraction ratio of the size in bits of the modulus p to the size in bits of the group order r. Recall that the number of points on the elliptic curve over the base field is given by $\#E = p + 1 - t$, where t is the trace of the Frobenius and $|t| \leq 2\sqrt{p}$, and $r|\#E$. The second parameter ω is the rounded-to-nearest-simple-fraction ratio of the size in bits of the order r to the size in bits of the trace t. In general we would prefer a small ρ, with $\rho = 1$ being regarded as ideal, and a large ω.

For example consider the BN family of curves [6], a family with an embedding degree of $k = 12$, and with a very simple generation function

$$p(x) = 36x^4 + 36x^3 + 24x^2 + 6x + 1$$

$$\#E(x) = 36x^4 + 36x^3 + 18x^2 + 6x + 1$$
$$t(x) = 6x^2 + 1$$

To find a pairing-friendly curve, simply choose an x of the appropriate size and check that $p(x)$ generates a prime. If $r(x) = \#E(x)$ is also prime, then in this case $\rho = 1$ and $\omega = 2$. The actual parameters of the curve can then be found using the method of Complex Multiplication [11]. Many other families of curves have also been discovered – see [16].

As a general truism it can be said that the closer ρ is to 1, the more difficult it becomes to force a low Hamming weight for r. On the other hand using a Cocks-Pinch pairing-friendly curve [7] for which $\rho = 2$, there is complete freedom in the choice of r. A value of $\rho = 1$ may however be mandatory in certain applications, for example short-signature schemes [9]. However it is our opinion that curves with $\rho = 2$ are still very usable (and of course there are a lot more of them), and their supposed inefficiency is often overstated (at higher levels of security in particular it is the extension field arithmetic which predominates in the calculation of the pairing, and it is the bit length of this extension field that matters, not the size of the base field from which it is constructed).

See below for the relevance of ω.

6 Optimizing the Miller loop

The algorithm as it currently stands looks like this – see algorithm 2.

Algorithm 2 Computation of $e(P, Q)$ with some optimizations

INPUT: $P \in E(\mathbb{F}_p), Q \in E(\mathbb{F}_{p^k})$, where P has order r
OUTPUT: $e(P, Q)$
1: $T \leftarrow P$, $f \leftarrow 1$
2: **for** $i \leftarrow \lfloor \lg(r) \rfloor - 1$ **downto** 0 **do**
3: $\quad f = f^2.l_{T,T}(Q)/v_{2T}(Q)$
4: $\quad T = 2T$
5: $\quad$ **if** $r_i = 1$ **then**
6: $\quad\quad f = f.l_{T,P}(Q)/v_{T+P}(Q)$
7: $\quad\quad T = T + P$
8: $\quad$ **end if**
9: **end for**
10: $f \leftarrow f^{p^d-1}$
11: $f \leftarrow f^{(p^d+1)/r}$
12: **return** f

The fact that k is always even allows us to assume that the extension field $\mathbb{F}_{p^k}$ is built as a quadratic extension on top of an implementation of $\mathbb{F}_{p^d}$. An element in $\mathbb{F}_{p^k}$ can then be represented as $w = a + ib$ where $a, b \in \mathbb{F}_{p^d}$. Then the conjugate of $w = \bar{w} = a - ib$. Now it is well known (Frobenius) that

$$(a+ib)^{p^d} = (a-ib)$$

The next optimization we suggest [23], [25] is to further exploit the fact that the output of the Miller loop is to be raised to the power of $p^d - 1$. From the above it follows immediately that

$$(1/(a+ib))^{p^d-1} = (a-ib)^{p^d-1}$$

In other words after raising to the power of $p^d - 1$ then the inverse and the conjugate cannot be distinguished. So now "push" the effect of the final exponentiation back into the main Miller loop, and replace inversions by much cheaper conjugations.

Algorithm 3 Computation of $e(P,Q)$ with further optimization

INPUT: $P \in E(\mathbb{F}_p), Q \in (\mathbb{F}_{p^k})$, where P has order r
OUTPUT: $e(P,Q)$

$T \leftarrow P$, $f \leftarrow 1$
for $i \leftarrow \lfloor \lg(r) \rfloor - 1$ **downto** 0 **do**
    $f = f^2.l_{T,T}(Q).\bar{v}_{2T}(Q)$
    $T = 2T$
    if $r_i = 1$ **then**
        $f = f.l_{T,P}(Q).\bar{v}_{T+P}(Q)$
        $T = T + P$
    end if
end for
$f \leftarrow f^{p^d-1}$
$f \leftarrow f^{(p^d+1)/r}$
return f

Now at a stroke we have gotten rid of the potentially very expensive extension field divisions in the main Miller loop (algorithm 3).

7 What about Q?

We have already chosen P to suit our needs, but what about Q? We would like to choose Q from a group disjoint from P, and ideally with some other advantages.

As things stand the point Q is a general point (x_Q, y_Q) on the elliptic curve over the field $\mathbb{F}_{p^k}$, where $x_Q = a + ib$ and $y_Q = c + id$, and $a, b, c, d \in \mathbb{F}_{p^d}$. Let us now restrict Q to be of a form where $b = c = 0$. This has an immediately useful effect as $\bar{v}_{2T}(Q)$ and $\bar{v}_{T+P}(Q)$ are now elements in the field $\mathbb{F}_{p^d}$, which means that they will be wiped out by the final exponentiation. This is the famous denominator-elimination optimization [3]. And as an extra bonus it is not difficult to establish that if $Q(a, id)$ is a point on $E(\mathbb{F}_{p^k})$, then it can be mapped to a point in an isomorphic group on the quadratic twist of this curve $E'(\mathbb{F}_{p^d})$.

So now Q, prior to its use in the pairing, can be manipulated as a point over the smaller extension field $\mathbb{F}_{p^d}$ [5] (requiring a small modification to the line function). See algorithm 4.

Algorithm 4 Computation of $e(P,Q)$ with denominator elimination

INPUT: $P \in E(\mathbb{F}_p), Q \in E'(\mathbb{F}_{p^d})$, where P has order r
OUTPUT: $e(P,Q)$
1: $T \leftarrow P$, $f \leftarrow 1$
2: **for** $i \leftarrow \lfloor \lg(r) \rfloor - 1$ **downto** 0 **do**
3: $\quad f = f^2.l_{T,T}(Q)$
4: $\quad T = 2T$
5: $\quad$ **if** $r_i = 1$ **then**
6: $\quad\quad f = f.l_{T,P}(T,Q)$
7: $\quad\quad T = T + P$
8: $\quad$ **end if**
9: **end for**
10: $f \leftarrow f^{p^d-1}$
11: $f \leftarrow f^{(p^d+1)/r}$
12: **return** f

In some situations and for certain types of pairings it may appear at first glance that such a favourable choice of Q may not be possible. However using a trick from [35] and [5], in fact it is always possible.

8 Some further optimizations

The group order r will always be odd, therefore the last iteration of the Miller loop will always invoke the $r = 1$ condition. However this last point addition, which takes T to the point at infinity, always results in a line value which will be again wiped out by the final exponentiation, and so this last step can be omitted [14].

Looking again at the final exponentiation we observe that in many cases the exponent p^d+1 can be further factored. For example $p^3+1 = (p+1)(p^2-p+1)$. From the definition of k, the group order must divide $p^2 - p + 1$, which is the sixth cyclotomic polynomial $\Phi_6(p)$. So in general the final exponentiation can be broken down into three parts, the easy exponentiation to the power of $p^d - 1$, the equally easy exponentiation to the power of $(p^d+1)/\Phi_k(p)$ (easy because the Frobenius can be used), and the "hard" exponentiation to the power of $\Phi_k(p)/r$. These modifications lead us to algorithm 5.

8.1 Calculating the hard part of the final exponentiation

When exponentiating in $\mathbb{F}_{p^k}$ there is never any need to use an exponent greater than p. Recall that exponentiation to the power of p is cheap, using the Frobenius.

Algorithm 5 Computation of $e(P,Q)$ with yet more optimization

INPUT: $P \in E(\mathbb{F}_p), Q \in E'(\mathbb{F}_{p^d})$, where P has order r
OUTPUT: $e(P,Q)$
1: $T \leftarrow P$, $f \leftarrow 1$
2: $s = r - 1$
3: **for** $i \leftarrow \lfloor \lg(s) \rfloor - 1$ **downto** 0 **do**
4: $\quad f = f^2.l_{T,T}(Q)$
5: $\quad T = 2T$
6: $\quad$ **if** $s_i = 1$ **then**
7: $\quad\quad f = f.l_{T,P}(Q)$
8: $\quad\quad T = T + P$
9: $\quad$ **end if**
10: **end for**
11: $f \leftarrow f^{p^d-1}$
12: $f \leftarrow f^{(p^d+1)/\Phi_k(p)}$
13: $f \leftarrow f^{\Phi_k(p)/r}$
14: **return** f

Therefore an exponent e can be represented to the base p as $e_0 + e_1.p + e_2.p^2 ....$ Now x^e can be written as

$$x^e = x^{e_0+e_1.p+e_2.p^2\cdots} = x^{e_0}.(x^p)^{e_1}.(x^{p^2})^{e_2}...$$

which can be quickly calculated using the Frobenius and the fast method of simultaneous exponentiation [22], [20], [31]. Standard windowing methods can be used, and the fact that inverses can be treated as conjugates can be further exploited to allow a NAF representation of the exponent.

For smaller values of $k \leq 8$, it may be faster to use Lucas or XTR style exponentiation [27], which uses the larger exponent, but only requires arithmetic over the smaller fields $\mathbb{F}_{p^{k/2}}$ or $\mathbb{F}_{p^{k/3}}$ respectively [40], [20]. As a nice side effect this has the result of "compressing" the value of the pairing to the smaller field size. Even if the multi-exponentiation method is used, it may still be useful to finally compress the pairing output.

9 Application-dependent Optimizations

It may be that in a particular application the first parameter P is a fixed constant, like perhaps a fixed private key. In this case it clearly makes sense to precompute the values of T which are just fixed multiples of P. In this situation it is always preferable to use affine coordinates for all the points (although in the general case where $P \in E(\mathbb{F}_p)$ projective coordinates are to be preferred).

Some protocols require the pairing value, an element of order r, to be subsequently further exponentiated to a value $v < r$. If using the multi-exponentiation method for the hard part of the final exponentiation, then we observe that

the further exponentiation to the power of v can be "folded into" the multi-exponentiation at virtually no extra cost. In many protocols these pairing exponentiations are included in the estimate of the protocols cost. Using this idea they can be obtained "for free". This trick works best for larger k.

If an entirely inversion-free pairing is desirable, then use projective coordinates throughout, and at little extra cost the full final exponentiation to the power of $(p^k - 1)/r$ can be carried out using one large multi-exponentiation.

10 Curve-dependent Optimizations

10.1 The Ate pairing

Some useful optimizations are possible, but they will depend on the particular type of pairing-friendly curve that is used. For example there are families of curves developed by Barreto et al. [4] and Duan et al. [13], for which the ω parameter is always greater than one. The same is true for the MNT curves where $\omega = 2$. In these cases a truncated loop variant of the Tate pairing is possible called the Ate pairing [24]. Here instead of putting the first parameter P on the curve over $\mathbb{F}_p$, and Q as a point on the twist over $\mathbb{F}_{p^d}$, we do it the other way around. Then, as it turns out, by simply substituting the line $s = t-1$ for the line $s = r - 1$ in algorithm 5, we still get a viable bilinear pairing (which has a simple relationship with the Tate pairing) with a Miller loop truncated by a factor of ω. For example for the BN curves mentioned above, the loop will be half the length of that required for the Tate pairing. Of course the fact that P is now taken over an extension field introduces an extra cost to the manipulation of T. This will not matter if P is fixed, as the precomputation optimization applies. In fact as we will see there are other ways to offset this extra cost.

Algorithm 6 Computation of $e(P, Q)$ with Ate pairing

INPUT: $P \in E'(\mathbb{F}_{p^d}), Q \in E(\mathbb{F}_p)$, P has order r
OUTPUT: $e(P, Q)$
1: $T \leftarrow P$, $f \leftarrow 1$
2: $s = t - 1$
3: **for** $i \leftarrow \lfloor \lg(s) \rfloor - 1$ **downto** 0 **do**
4: $\quad f = f^2.l_{T,T}(Q)$
5: $\quad T = 2T$
6: $\quad$ **if** $s_i = 1$ **then**
7: $\quad\quad f = f.l_{T,P}(Q)$
8: $\quad\quad T = T + P$
9: $\quad$ **end if**
10: **end for**
11: $f \leftarrow f^{p^d-1}$
12: $f \leftarrow f^{(p^d+1)/\Phi_k(p)}$
13: $f \leftarrow f^{\Phi_k(p)/r}$
14: **return** f

10.2 Low Discriminant Complex Multiplication Curves

The parameters of all non-supersingular pairing-friendly elliptic curves must be found using the method of Complex Multiplication (CM method) [11]. Many families of pairing-friendly curves are found to have a CM discriminant of -1 or -3 [4], [13], [6]. In these cases the curves exhibit quartic and sextic twists respectively, as well as quadratic twists. Now it may be possible to select Q in the Tate pairing (and P in the Ate pairing) from a higher order twist, and hence over a lower order field. For example for the BN curves for which $k = 12$ it is possible to place P on $\mathbb{F}_p$ and Q on the sextic twist over $\mathbb{F}_{p^2}$, or vice versa for the Ate pairing. In fact for a $k = 6$ pairing friendly curve with a CM discriminant of -3, it is perfectly possible to have both P and Q as points on curves over $\mathbb{F}_p$. However we are only able to find such a pairing friendly curve with $\rho = 2$. For example consider this family of curves [16].

$$\begin{aligned} p(x) &= 27x^4 + 9x^3 + 3x^2 + 3x + 1 \\ r(x) &= 9x^2 + 3x + 1 \\ t(x) &= 3x + 2 \end{aligned}$$

Observe that $\omega = 2$, so the Ate pairing will be very efficient in this case. And don't be put off by $\rho = 2$ – this is a useful curve!

In all these cases the line functions $l_{A,B}(Q)$ will be of a sparse form which will further speed up calculations. An alternative idea to exploit low discriminant curves, which is particularly effective for the case $k = 2$, is described in [37].

10.3 Truncated final exponentiation

In some cases the method of generation of the curves allows us to dramatically shorten the hard part of the final exponentiation. Consider for example a $k = 6$ MNT curve. In this case the exponent will be $(p^2 - p + 1)/r$. Assume that $r = p + 1 - t$, in other words the number of points on the base field curve is a prime. Then this exponent will be $(p^2 - p + 1)/(p + 1 - t) = p \pm \delta$, where $\delta \approx t$. Therefore the hard part of the final exponentiation will be $f^{p \pm \delta} = f^p . f^{\pm \delta}$, which can be calculated using a Frobenius and a half-length exponentiation (and avoiding multi-exponentiation). This same idea applies to a varying extent to other families of curves – for example for the BN curves the hard part of the final exponentiation requires only a three-quarters length exponent.

10.4 Avoiding multi-exponentiation

Using multi-exponentiation for the final exponentiation is fast, but requires a lot of memory for precomputed values. As it happens, in many cases it can be avoided. Considering again the final exponentiation for the BN curves, it is not hard to work out that the "hard part" of the final exponentiation in algorithm 6 can be calculated using multi-exponentiation as

$$f \leftarrow f^{p^3}.(f^{p^2})^{6x^2+1}.(f^p)^{36x^3-18x^2+12x+1}.f^{36x^3-30x^2+18x-2}$$

But with a little work this can also be equivalently calculated as

$$\begin{aligned} a &= f^{6x-5} \\ b &= a^p \\ c &= ab \\ f &\leftarrow f^{p^3}.[c.(f^p)^2.f^{p^2}]^{6x^2+1}.c.(f^p.f)^9.a.f^4 \end{aligned}$$

requiring only simple exponentiation and some multiplications and squarings. For x negative, the first step can be calculated instead as $a = 1/f^{5-6*x}$. Note that the overall length of exponent in bits is not any greater than for the multi-exponentiation. Note also that if x has a low Hamming weight, windowing methods of exponentiation will not be much better than a simple square-and-multiply algorithm, again saving on memory for precomputed values. This could be useful in a constrained environment. In some experiments this method was found to be about 20% faster than using multi-exponentiation.

10.5 Super pairing-friendly curves

Using the method of Barreto, Lynn and Scott [4], we can find this pairing-friendly curve:-

$$\begin{aligned} D &= -1 \\ p(x) &= (1 + 2x + x^2 + x^6 - 2x^7 + x^8)/4 \\ t(x) &= x + 1 \\ \#E(x) &= ((x-1)^2)(x^2+1)(x^4 - x^2 + 1)/4 \\ &= ((x-1)^2.\Phi_4(x).\Phi_{12}(x))/4 \end{aligned}$$

We define a curve whose number of points is divisible by more than one cyclotomic polynomial, as a "super pairing-friendly" curve.

By choosing points of an order which divides $\Phi_4(x)$ we can do pairing-based cryptography with an embedding degree of 4, albeit with a rather high ρ value of 4. Using points of an order which divides $\Phi_{12}(x)$ we can do pairing-based cryptography *on the same curve* with an embedding degree of 12 and $\rho = 2$.

One might for example choose x of about 128 bits (taking care that both $\Phi_4(x)$ and $\Phi_{12}(x)$ evaluate as primes or near-primes), to achieve a security equivalent to 128-bit AES, with a prime modulus of 1024 bits, using the embedding degree of 4. The discrete logarithm difficulty would be equivalent to about 4.1024 = 4096 bits, which is about right. Later one could switch to an embedding degree of 12 on the same curve, to obtain security roughly equivalent to 256-bit AES, and with a discrete logarithm difficulty of 12.1024 = 12288 bits, which again is about right.

11 The wider context

A pairing is not calculated in isolation, it is calculated as part of a wider context, to implement some useful cryptographic protocol.

11.1 Which embedding degree?

One question that arises immediately is which embedding degree should be used for a particular level of security. The answer is, regrettably, that it depends. Our view on it is summarised in Table 1.

Table 1. Key size security in bits

Symmetric key size	Group size	Extension field size	Embedding degree
80	160	960 – 1280	2–8
128	256	3000 – 5000	12–18
256	512	12000 – 18000	24–36

For example at the 80-bit level of security, one could use a $k = 2$ Cocks-Pinch curve with a 512-bit prime p, for an extension field size of 1024-bits, and using a group size r of 160 bits. Alternatively one could use a $k = 6$ MNT curve again with a 160-bit prime p and group size of 160 bits, and an extension field size of 960 bits. On the face of it these choices represent similar levels of security. However in our experience the former is somewhat faster when it comes to calculating the pairing. On the other hand the protocol may require, as well as the pairing, point multiplications, and in this case the smaller size of p (the field over which the pairing-friendly elliptic curve will be defined) will be a big advantage. On the other other hand these point multiplications may be of fixed points, in which case, using precomputation, the cost of point multiplication may in fact be insignificant, as in the case of Boneh and Franklin IBE [8]. For the $k = 2$ curve, precomputation based on a fixed point P will be a big advantage, much less so on the $k = 6$ curve. For the $k = 2$ curve the parameter Q will be on the twist also over $\mathbb{F}_p$, whereas for the $k = 6$ MNT curve the twist is over $\mathbb{F}_{p^3}$, which is more complicated. If one was implementing a short-signature scheme [9], then the $k = 2$ curve cannot even be considered. I could go on....

The issue of how best to scale security in pairings has been much debated, see [26], [20], [38]. One point perhaps not considered is that the code for higher extension fields becomes a lot "fussier", in that rather than spending most of its time for example in the inner loops of a modular multiplication, it spends more time hopping around in and out of functions, and accumulating costs from function overheads which can be quite significant. We have already observed how in a constrained cache environment such code can be punished by an increased rate of instruction cache misses [41].

11.2 Products of pairings

In the case of a protocol which requires the products of pairings, three ideas can be used to speed up the calculation [36]. For example consider the calculation of $e(P,Q) \cdot e(R,S)$. These can be combined into a single algorithm.

- Since the implicit multiplications of P and R occur in lock-step with one another, it makes sense to use affine coordinates in conjunction with Montgomery's trick. This means that just one modular inversion will be required instead of two. Montgomery's trick is based on the simple observation that $1/x = y/xy$ and $1/y = x/xy$.
- Both pairings can share the same *Miller variable* f. This means only a single squaring of f in the combined pairing algorithm will be required.
- Both pairings can share the final exponentiation (as pointed out by Solinas [42]).

These ideas are further evaluated in [21].

12 Timings

Here we present some timings, which, at the 80-bit level of security compare three quite different pairing implementations, and compare them with a standard 1024-bit RSA decryption on the same platform, a standard PC. All the code is written in C plus some assembly language. The Tate pairing implementation uses a $k = 2$ Cocks-Pinch curve, as described above. The Ate pairing implementation uses a $k = 4$ pairing-friendly curve with $\omega = 2$, a group size of 160-bits and a prime modulus of 256 bits, with an extension field size of $4.256 = 1024$ bits. These pairings are calculated in a context where precomputation on the first pairing parameter P is applicable. Finally these are compared with an implementation of the η_T pairing [2], which uses a standard supersingular elliptic curve of characteristic 2, with an embedding degree of $k = 4$. The point multiplication timings are for the multiplication of a variable point on the elliptic curve over the base field by a random 160-bit value. The field exponentiation, typically of a pairing value, uses a random 160-bit exponent.

Table 2. Timings in milliseconds on 3GHz Pentium IV

	$E(\mathbb{F}_{2^{379}})$ η_T pairing	$E(\mathbb{F}_p)$ Tate pairing	$E(\mathbb{F}_p)$ Ate pairing
Pairing	3.88	2.91	3.10
Point Mult.	1.82	3.08	1.17
Field exp.	1.14	0.54	0.62
RSA decryption	1.92		

References

1. D. Bailey and C. Paar. Optimal extension field for fast arithmetic in public key algorithms. In *Advances in Cryptology – Crypto'1998*, volume 1462 of *Lecture Notes in Computer Science*, pages 472–485. Springer-Verlag, 1998.
2. P. S. L. M. Barreto, S. Galbraith, C. O'hEigeartaigh, and M. Scott. Efficient pairing computation on supersingular abelian varieties. Cryptology ePrint Archive, Report 2004/375, 2004. `http://eprint.iacr.org/2004/375`.
3. 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.
4. P. S. L. M. Barreto, B. Lynn, and M. Scott. Constructing elliptic curves with prescribed embedding degrees. In *Security in Communication Networks – SCN'2002*, volume 2576 of *Lecture Notes in Computer Science*, pages 263–273. Springer-Verlag, 2002.
5. Paulo 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, Ottawa, Canada, 2003. Springer-Verlag.
6. P.S.L.M. Barreto and M. Naehrig. Pairing-friendly elliptic curves of prime order. Cryptology ePrint Archive, Report 2005/133, 2005. `http://eprint.iacr.org/2005/133`.
7. I. F. Blake, G. Seroussi, and N. P. Smart, editors. *Advances in Elliptic Curve Cryptography, Volume 2*. Cambridge University Press, 2005.
8. D. Boneh and M. Franklin. Identity-based encryption from the Weil pairing. *SIAM Journal of Computing*, 32(3):586–615, 2003.
9. D. Boneh, B. Lynn, and H. Shacham. Short signatures from the Weil pairing. In *Advances in Cryptology – Asiacrypt'2001*, volume 2248 of *Lecture Notes in Computer Science*, pages 514–532. Springer-Verlag, 2002.
10. J. Chung and M. A. Hasan. Asymmetric squaring formulae, 2006. `http://www.cacr.math.uwaterloo.ca/`.
11. R. Crandall and C. Pomerance. *Prime Numbers: a Computational Perspective*. Springer-Verlag, Berlin, 2001.
12. R. Dahab, A. J. Devegili, C. OhEigeartaigh, and M. Scott. Multiplication and squaring on pairing-friendly fields. Cryptology ePrint Archive, Report 2006/471, 2006. `http://eprint.iacr.org/2006/471`.
13. P. Duan, S. Cui, and C. Wah Chan. Special polynomial families for generating more suitable elliptic curves for pairing-based cryptosystems. Cryptology ePrint Archive, Report 2005/342, 2006. `http://eprint.iacr.org/2005/342`.
14. 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.
15. K. Eisentrager, K. Lauter, and P. L. Montgomery. Fast elliptic curve arithmetic and improved weil pairing evaluation. In *Topics in Cryptology, CT-RSA03*, volume 2612 of *Lecture Notes in Computer Science*, pages 343–354. Springer-Verlag, 2003.
16. D. Freeman, M. Scott, and E. Teske. A taxonomy of pairing-friendly elliptic curves. Cryptology ePrint Archive, Report 2006/372, 2006. `http://eprint.iacr.org/2006/372`.
17. G. Frey, M. Müller, and H. Rück. The Tate pairing and the discrete logarithm applied to elliptic curve cryptosystems. *IEEE Transactions on Information Theory*, 45(5):1717–1719, 1999.

18. P. Friedland. Absolute value and square root of a complex number. *Communications of the ACM*, 10:665, 1967.
19. S. Galbraith, K. Harrison, and D. Soldera. Implementing the Tate pairing. In *Algorithm Number Theory Symposium – ANTS V*, volume 2369 of *Lecture Notes in Computer Science*, pages 324–337. Springer-Verlag, 2002.
20. R. Granger, D. Page, and N. P. Smart. High security pairing-based cryptography revisited. In *Algorithmic Number Theory Symposium – ANTS VII*, volume 4076 of *Lecture Notes in Computer Science*, pages 480–494. Springer-Verlag, 2006.
21. R. Granger and N. P. Smart. On computing products of pairings. Cryptology ePrint Archive, Report 2006/172, 2006. `http://eprint.iacr.org/2006/172`.
22. L. Hei, J. Dong, and D. Pei. Implementation of cryptosystems based on Tate pairing. *J. Comput. Sci & Technolgy*, 20(2):264–269, 2005.
23. C. Ó hÉigeartaigh. Speeding up pairing computation, 2005. `http://eprint.iacr.org/2005/293`.
24. F. Hess, N. Smart, and F. Vercauteren. The Eta pairing revisited. Cryptology ePrint Archive, Report 2006/110, 2006. `http://eprint.iacr.org/2006/110`.
25. T. Kobayashi, K. Aoki, and H. Imai. Efficient algorithms for Tate pairing. *IEICE Trans. Fundamentals*, E89-A, 2006.
26. N. Koblitz and A. Menezes. Pairing-based cryptography at high security levels. In *Cryptography and Coding: 10th IMA International Conference*, volume 3796 of *Lecture Notes in Computer Science*, pages 13–36. Springer-Verlag, 2005.
27. A. Lenstra and E. Verheul. An overview of the XTR public key system. In *Warsaw Conference on Public-Key cryptography and computational number theory*, 2001.
28. C. H. Lim and H. S. Hwang. Fast implementation of elliptic curve arithmetic in GF(p^n). In *Proceedings of PKC'2000*, volume 1751 of *Lecture Notes in Computer Science*, pages 405–421. Springer-Verlag, 2000.
29. Ben Lynn. PBC library, 2006. `http://rooster.stanford.edu/~ben/pbc/download.html`.
30. A. Menezes. *Elliptic Curve Public Key Cryptosystems*. Kluwer Academic Publishers, 1993.
31. Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of applied cryptography*. CRC Press, Boca Raton, Florida, 1996. URL: `http://cacr.math.uwaterloo.ca/hac`.
32. A. Miyaji, M. Nakabayashi, and S. Takano. New explicit conditions of elliptic curve traces for FR-reduction. *IEICE Transactions on Fundamentals*, E84-A(5):1234–1243, 2001.
33. P. Montgomery. Modular multiplication without division. *Mathematics of Computation*, 44(170):519–521, 1985.
34. O. Schirokauer. The number field sieve for integers of low weight. Cryptography ePrint Archive, Report 2006/107, 2006. `http://eprint.iacr.org/2006/107`.
35. M. Scott. Faster identity based encryption. *Electronics Letters*, 40(14):861, 2004.
36. M. Scott. Computing the Tate pairing. In *CT-RSA*, volume 3376 of *Lecture Notes in Computer Science*, pages 293–304. Springer-Verlag, 2005.
37. M. Scott. A note on Boneh and Franklin IBE, 2005. `ftp://ftp.computing.dcu.ie/pub/resources/crypto/note.pdf`.
38. M. Scott. Scaling security in pairing-based protocols. Cryptology ePrint Archive, Report 2005/139, 2005. `http://eprint.iacr.org/139`.
39. M. Scott, 2006. `http://ftp.computing.dcu.ie/pub/crypto/miracl.zip`.
40. 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/`.

41. M. Scott, N. Costigan, and W. Abdulwahab. Implementing cryptographic pairings on smartcards, 2006. http://eprint.iacr.org/2006/144.
42. J. Solinas. ID-based digital signature algorithms, 2003. http://www.cacr.math.uwaterloo.ca/conferences/2003/ecc2003/solinas.pdf.
43. F. Wang, Y. Nogami, and Y. Morikawa. An efficient square root computation in finite fields GF(p^{2^d}). *IEICE Trans. Fundamentals*, E88-A(10):2792–2799, 2005.