An Analysis of Affine Coordinates for Pairing Computation

Kristin Lauter, Peter L. Montgomery, and Michael Naehrig

Microsoft Research, One Microsoft Way, Redmond, WA 98052, USA
{klauter, petmon, mnaehrig}@microsoft.com

Abstract. In this paper we analyze the use of affine coordinates for pairing computation. We observe that in many practical settings, e. g. when implementing optimal ate pairings in high security levels, affine coordinates are faster than using the best currently known formulas for projective coordinates. This observation relies on two known techniques for speeding up field inversions which we analyze in the context of pairing computation. We give detailed performance numbers for a pairing implementation based on these ideas, including timings for base field and extension field arithmetic with relative ratios for inversion-to-multiplication costs, timings for pairings in both affine and projective coordinates, and average timings for multiple pairings and products of pairings.
Keywords: Pairing computation, affine coordinates, optimal ate pairing, finite field inversions, pairing cost, multiple pairings, pairing products.

1 Introduction

Cryptographic pairing computations are required for a wide variety of new cryptographic protocols and applications. All cryptographic pairings currently used in practice are based on pairings on elliptic curves, requiring both elliptic curve operations and function computation and evaluation to compute the pairing of two points on an elliptic curve [36]. For a given security level, it is important to optimize efficiency of the pairing computation, and much work has been done on this topic (see for example [6,7,5,30,35,44,42]).

Elliptic curve operations can be implemented using various coordinate systems, such as affine or different variants of projective coordinates (for an overview see [10]). It has long been the case that many implementers have found affine coordinates slow for elliptic curve operations because of the relatively high costs of inversions and the relatively fast modular multiplication that can be achieved for special moduli such as generalized Mersenne primes. Thus projective coordinates were also suggested for pairing implementations [33,41], and very efficient explicit formulas were found for various parameter choices [1,16]. So recently there has been a bias in the literature towards the use of projective coordinates for pairings as well. Nevertheless, researchers had previously concluded that affine coordinates can be superior in many situations (see [22, Section 5] and [21, Section IX.14]).

In this paper we analyze the use of affine coordinates for pairing computation in different settings. We propose the use of two known techniques for speeding up field inversions and analyze them in the context of pairing computation. Based on these, we find that in many practical settings, for example when implementing one of the optimal pairings [44] based on the ate pairing [30] in high security levels, affine coordinates will be much faster than projective coordinates.

The first technique we investigate is computing inverses in extension fields by using towers of extension fields and successively reducing inverse computation to subfield computations via the norm map. We show that this technique drastically reduces the ratio of the costs of inversions to multiplications in extension fields. Thus when computing the ate pairing, where most computations take place in a potentially large extension field, the advantage of projective coordinates is eventually erased as the degree of the extension gets large. This happens for example when implementing pairings on curves for higher security levels such as 256 bits, or when special high-degree twists can not be used to reduce the size of the extension field.

The second technique we investigate is the use of inversion-sharing for pairing computations. Inversion-sharing is a standard trick whenever several inversions are computed at once. As the number of elements to be inverted grows, the average ratio of inversion-to-multiplication costs approaches 3. Inversion-sharing can be used in a single pairing computation if the binary expansion is read from right-to-left instead of left-to-right. This approach also has the advantage that it can be easily parallelized to take advantage of multi-core processors. Inversion-sharing for pairing computation can also be advantageous for computing multiple pairings or for computing products of pairings, as was suggested by Scott [41] and analyzed by Granger and Smart [25].

Ironically, although the two techniques we investigate can be used simultaneously, it is often not necessary to do so, since either technique alone can reduce the inversion to multiplication ratio. Either technique alone makes affine coordinates faster than projective coordinates in some settings.

To illustrate these techniques, we give detailed performance numbers for a pairing implementation based on these ideas. This includes timings for base field and extension field arithmetic with relative ratios for inversion-to-multiplication costs and timings for pairings in both affine and projective coordinates, as well as average timings for multiple pairings and products of pairings. In our implementation, affine coordinates are faster than projective coordinates even for Barreto-Naehrig curves [8] with a high-degree twist at the lowest security levels. However, we expect that for other implementations, the benefits of affine coordinates would only be realized for higher security levels or for curves without high-degree twists.

The paper is organized as follows: Section 2 provides the necessary background on the ate pairing and discusses the costs of doubling and addition steps in Miller's algorithm. In Section 3, we show how variants of the ate pairing can benefit from using affine coordinates due to the fact that the inversion-to-multiplication ratio in an extension field is much smaller than in the base field.

Section 4 is dedicated to revisiting the well-known inversion-sharing trick and its application in pairing computation. Finally, Section 5 gives benchmarking results for our pairing implementation based on the Microsoft Research bignum library.

2 Pairing computation

Let $p > 3$ be a prime and $\mathbb{F}_q$ be a finite field of characteristic p. Let E be an elliptic curve defined over $\mathbb{F}_q$, given by $E : y^2 = x^3 + ax + b$, where $a, b \in \mathbb{F}_q$ and $4a^3 + 27b^2 \neq 0$. We denote by $\mathcal{O}$ the point at infinity on E. Let $n = \#E(\mathbb{F}_q) = q + 1 - t$, where t is the trace of Frobenius, which fulfills $|t| \leq 2\sqrt{q}$. We fix a prime r with $r \mid n$. Let k be the embedding degree of E with respect to r, i. e. k is the smallest positive integer with $r \mid q^k - 1$. This means that $\mathbb{F}_{q^k}^*$ contains the group μ_r of r-th roots of unity. The embedding degree of E is an important parameter, since it determines the field extensions over which the groups that are involved in pairing computation are defined.

For $m \in \mathbb{Z}$, let $[m]$ be the multiplication-by-m map. The kernel of $[m]$ is the set of m-torsion points on E; it is denoted by $E[m]$ and we write $E(\mathbb{F}_{q^\ell})[m]$ for the set of $\mathbb{F}_{q^\ell}$-rational m-torsion points ($\ell > 0$). If $k > 1$, which we assume from now on, we have $E[r] \subseteq E(\mathbb{F}_{q^k})$, i.e. all r-torsion points are defined over $\mathbb{F}_{q^k}$.

Most pairings that are suitable for use in practical cryptographic applications are derived from the Tate pairing, which is a map $E(\mathbb{F}_{q^k})[r] \times E(\mathbb{F}_{q^k})/rE(\mathbb{F}_{q^k}) \to \mathbb{F}_{q^k}^*/(\mathbb{F}_{q^k}^*)^r$ (for details see [18,19]). In this paper, we focus on the ate pairing [30], variants of which are often the most efficient choices for implementation.

2.1 The ate pairing

Given $m \in \mathbb{Z}$ and $P \in E[r]$, let $f_{m,P}$ be a rational function on E with divisor $(f_{m,P}) = m(P) - ([m]P) - (m-1)(\mathcal{O})$. Let ϕ_q be the q-power Frobenius endomorphism on E. Define two groups of prime order r by $G_1 = E[r] \cap \ker(\phi_q - [1]) = E(\mathbb{F}_q)[r]$ and $G_2 = E[r] \cap \ker(\phi_q - [q]) \subseteq E(\mathbb{F}_{q^k})[r]$. The *ate pairing* is defined as

$$a_T : G_2 \times G_1 \to \mu_r, \quad (Q, P) \mapsto f_{T,Q}(P)^{(q^k-1)/r}, \tag{1}$$

where $T = t - 1$. The group G_2 has a nice representation by an isomorphic group of points on a twist E' of E, which is a curve that is isomorphic to E. Here, we are interested in those twists which are defined over a subfield of $\mathbb{F}_{q^k}$ such that the twisting isomorphism is defined over $\mathbb{F}_{q^k}$. Such a twist E' of E is given by an equation $E' : y^2 = x^3 + (a/\alpha^4)x + (b/\alpha^6)$ for some $\alpha \in \mathbb{F}_{q^k}$ with isomorphism $\psi : E' \to E,\ (x, y) \mapsto (\alpha^2 x, \alpha^3 y)$. If ψ is minimally defined over $\mathbb{F}_{q^k}$ and E' is minimally defined over $\mathbb{F}_{q^{k/d}}$ for a $d \mid k$, then we say that E' is a twist of degree d. If $a = 0$, let $d_0 = 4$; if $b = 0$, let $d_0 = 6$, and let $d_0 = 2$ otherwise. For $d = \gcd(k, d_0)$ there exists exactly one twist E' of E of degree d for which $r \mid \#E'(\mathbb{F}_{q^{k/d}})$ (see [30]). Define $G_2' = E'(\mathbb{F}_{q^{k/d}})[r]$. Then the map ψ is

a group isomorphism $G_2' \to G_2$ and we can represent all elements in G_2 by the corresponding preimages in G_2'. Likewise, all arithmetic that needs to be done in G_2 can be carried out in G_2'. The advantage of this is that points in G_2' are defined over a smaller field than those in G_2. Using G_2', we may now view the ate pairing as a map $G_2' \times G_1 \to \mu_r,\ (Q', P) \mapsto f_{T,\psi(Q')}(P)^{(q^k-1)/r}$.

The computation of $a_T(Q', P)$ is done in two parts: first the evaluation of the function $f_{T,\psi(Q')}$ at P, and second the so-called final exponentiation to the power $(q^k - 1)/r$. The first part is done with Miller's algorithm [36]. We describe it for even embedding degree in Algorithm 1 which shows how to compute $f_{m,\psi(Q')}(P)$ for some integer $m > 0$. We denote the function given by the line through two points R_1 and R_2 on E by l_{R_1,R_2}. If $R_1 = R_2$, then the line is given by the tangent to the curve passing through R_1. Throughout this paper, we assume that k is even so that denominator elimination techniques can be used (see [6,7]).

Algorithm 1 Miller's algorithm for even k and ate-like pairings

Input: $Q' \in G_2', P \in G_1, m = (1, m_{l-2}, \ldots, m_0)_2$
Output: $f_{m,\psi(Q')}(P)$ representing a class in $\mathbb{F}_{q^k}^*/(\mathbb{F}_{q^k}^*)^r$

1: $R' \leftarrow Q', \quad f \leftarrow 1$
2: **for** i from $\ell - 1$ downto 0 **do**
3: $\quad f \leftarrow f^2 \cdot l_{\psi(R'),\psi(R')}(P), \quad R' \leftarrow [2]R'$
4: $\quad$ **if** $(m_i = 1)$ **then**
5: $\quad\quad f \leftarrow f \cdot l_{\psi(R'),\psi(Q')}(P), \quad R' \leftarrow R' + Q'$
6: $\quad$ **end if**
7: **end for**
8: **return** f

Miller's algorithm builds up the function value $f_{m,\psi(Q')}(P)$ in a square-and-multiply-like fashion from line function values along a scalar multiplication computing $[m]Q'$ (which is the value of R' after the Miller loop). Step 3 is called a *doubling step*, it consists of squaring the intermediate value $f \in \mathbb{F}_{q^k}$, multiplying it with the function value given by the tangent to E in $R = \psi(R')$, and doubling the point R'. Similarly, an *addition step* is computed in Step 5 of Algorithm 1.

The final exponentiation in (1) maps classes in $\mathbb{F}_{q^k}^*/(\mathbb{F}_{q^k}^*)^r$ to unique representatives in μ_r. Given the fixed special exponent, there are many techniques that improve its efficiency significantly over a plain exponentiation (see [42,24]).

The most efficient variants of the ate pairing are so-called optimal ate pairings [44]. They are optimal in the sense that they minimize the size of m and with that the number of iterations in Miller's algorithm to $\log(r)/\varphi(k)$, where φ is the Euler totient function. For these minimal values of m, the function $f_{m,\psi(Q')}$ alone usually does not give a bilinear map. To get a pairing, these functions need to be adjusted by multiplying with a small number of line function values; for details we refer to [44].

Secure and efficient implementation of pairings can be done only with a careful choice of the underlying elliptic curve. The curve needs to be pairing-

friendly, i.e. the embedding degree k needs to be small, while r should be larger than $\sqrt{q}$. A survey of methods to construct such curves can be found in [20]. For security reasons, the parameters need to have certain minimal sizes which lead to optimal values for the embedding degree k for specific security levels (see for example the keysize recommendations in [43] and [4]).

Furthermore, it is advantageous to choose curves with twists of degree 4 or 6, so-called high-degree twists, since this results in higher efficiency due to the more compact representation of the group G_2. To achieve security levels of 128 bits or higher, embedding degrees of 12 and larger are optimal. Because the degree of the twist E' is at most 6, this means that when computing ate-like pairings at such security levels, all field arithmetic in the doubling and addition steps in Miller's algorithm takes place over a proper extension field of $\mathbb{F}_q$.

2.2 Costs for doubling and addition steps

In this section, we take a closer look at the costs of the doubling and addition steps in Miller's algorithm. We begin by describing the evaluation of line functions in affine coordinates, i.e. a point P on E, $P \neq \mathcal{O}$, is given by two affine coordinates as $P = (x_P, y_P)$. Let $R_1, R_2, S \in E$ with $R_1 \neq -R_2$ and $R_1, R_2 \neq \mathcal{O}$. Then the function of the line through R_1 and R_2 (tangent to E if $R_1 = R_2$) evaluated at S is given by $l_{R_1,R_2}(S) = y_S - y_{R_1} - \lambda(x_S - x_{R_1})$, where $\lambda = (3x_{R_1}^2 + a)/2y_{R_1}$ if $R_1 = R_2$ and $\lambda = (y_{R_2} - y_{R_1})/(x_{R_2} - x_{R_1})$ otherwise. The value λ is also used to compute $R_3 = R_1 + R_2$ on E by $x_{R_3} = \lambda^2 - x_{R_1} - x_{R_2}$ and $y_{R_3} = \lambda(x_{R_1} - x_{R_3}) - y_{R_1}$. If $R_1 = -R_2$, then we have $x_{R_1} = x_{R_2}$ and $l_{R_1,R_2}(S) = x_S - x_{R_1}$.

Before analyzing the costs for doubling and addition steps, we introduce notations for field arithmetic costs. Let $\mathbb{F}_{q^m}$ be an extension of degree m of $\mathbb{F}_q$ for $m \geq 1$. We denote by $\mathbf{M}_{q^m}$, $\mathbf{S}_{q^m}$, $\mathbf{I}_{q^m}$, $\mathbf{add}_{q^m}$, $\mathbf{sub}_{q^m}$, and $\mathbf{neg}_{q^m}$ the costs for multiplication, squaring, inversion, addition, subtraction, and negation in the field $\mathbb{F}_{q^m}$. When we omit the indices in all of the above, this indicates that we are dealing with arithmetic in a fixed field and field extensions do not play a role. The cost for a multiplication by a constant $\omega \in \mathbb{F}_{q^m}$ is denoted by $\mathbf{M}_{(\omega)}$. We assume the same costs for addition of a constant as for a general addition.

Let notations be as described in Section 2.1. Let $e = k/d$, then $G_2' = E'(\mathbb{F}_{q^e})[r]$. Let $P \in G_1$, $R', Q' \in G_2'$ and let $R = \psi(R'), Q = \psi(Q')$. Furthermore, we assume that $\mathbb{F}_{q^k} = \mathbb{F}_{q^e}(\alpha)$ where $\alpha \in \mathbb{F}_{q^k}$ is the same element as the one defining the twist E', and we have $\alpha^d = \omega \in \mathbb{F}_{q^e}$. This means that each element in $\mathbb{F}_{q^k}$ is given by a polynomial of degree $d-1$ in α with coefficients in $\mathbb{F}_{q^e}$ and the twisting isomorphism ψ maps (x', y') to $(\alpha^2 x', \alpha^3 y')$.

Doubling steps in affine coordinates: We need to compute

$$l_{R,R}(P) = y_P - \alpha^3 y_{R'} - \lambda(x_P - \alpha^2 x_{R'}) = y_P - \alpha\lambda' x_P + \alpha^3(\lambda' x_{R'} - y_{R'})$$

and $R_3' = [2]R'$, where $x_{R_3'} = \lambda'^2 - 2x_{R'}$ and $y_{R_3'} = \lambda'(x_{R'} - x_{R_3'}) - y_{R'}$. We have $\lambda' = (3x_{R'}^2 + a/\alpha^4)/2y_{R'}$ and $\lambda = (3x_R^2 + a)/2y_R = \alpha\lambda'$. Note that $[2]R' \neq \mathcal{O}$ in the pairing computation.

The slope λ' can be computed with $\mathbf{I}_{q^e}+\mathbf{M}_{q^e}+\mathbf{S}_{q^e}+4\mathbf{add}_{q^e}$, assuming that we compute $3x_{R'}^2$ and $2y_{R'}$ by additions. To compute the double of R' from the slope λ', we need at most $\mathbf{M}_{q^e}+\mathbf{S}_{q^e}+4\mathbf{sub}_{q^e}$. We obtain the line function value with a cost of $e\mathbf{M}_q$ to compute $\lambda' x_P$ and $\mathbf{M}_{q^e}+\mathbf{sub}_{q^e}+\mathbf{neg}_{q^e}$ for $d \in \{4,6\}$. When $d = 2$, note that $\alpha^2 = \omega \in \mathbb{F}_{q^e}$ and thus we need $(k/2)\mathbf{M}_q+\mathbf{M}_{q^{k/2}}+\mathbf{M}_{(\omega)}+2\mathbf{sub}_{q^{k/2}}$ for the line.

We summarize the operation counts in Table 1. We restrict to even embedding degree and $4 \mid k$ for $b = 0$ as well as to $6 \mid k$ for $a = 0$ because these cases allow using the maximal-degree twists and are likely to be used in practice. We compare the affine counts to costs of the fastest known formulas using projective coordinates taken from [31] and [16]; see these papers for details. For an overview of the most efficient explicit formulas known for elliptic-curve operations see the EFD [10]. We transfer the formulas in [31] to the ate pairing using the trick in [16] where the ate pairing is computed entirely on the twist. In this setting we assume field extensions are constructed in a way that favors the representation of line function values. This means that the twist isomorphism can be different from the one described in this paper. Still, in the case $d = 2$, evaluation of the line function can not be done in $k\mathbf{M}_q$; instead 2 multiplications in $\mathbb{F}_{q^{k/2}}$ need to be done (see [16]). Furthermore, we assume that all precomputations are done as described in the above papers and small multiples are computed by additions.

DBL	d	coord.	$\mathbf{M}_q$	$\mathbf{I}_{q^e}$	$\mathbf{M}_{q^e}$	$\mathbf{S}_{q^e}$	$\mathbf{M}_{(\ .\ )}$	$\mathbf{add}_{q^e}$	$\mathbf{sub}_{q^e}$	$\mathbf{neg}_{q^e}$
$ab \neq 0$	2	affine	$k/2$	1	3	2	$1\mathbf{M}_{(\omega)}$	4	6	–
$2 \mid k$		Jac. [31]	–	–	3	11	$1\mathbf{M}_{(a/\omega^2)}$	6	17	–
$b = 0$	4	affine	$k/4$	1	3	2	–	4	5	1
$4 \mid k$		$W_{(1,2)}$ [16]	$k/2$	–	2	8	$1\mathbf{M}_{(a/\omega)}$	9	10	1
$a = 0$	6	affine	$k/6$	1	3	2	–	4	5	1
$6 \mid k$		proj. [16]	$k/3$	–	2	7	$1\mathbf{M}_{(b/\omega)}$	11	10	1

Table 1. Operation counts for the doubling step in the ate-like Miller loop omitting $1\mathbf{S}_{q^k} + 1\mathbf{M}_{q^k}$.

Addition steps in affine coordinates: The line function value has the same shape as for doubling steps. Note that we can replace R' by Q' in the line and compute

$$l_{R,Q}(P) = y_P - \alpha^3 y_{Q'} - \lambda(x_P - \alpha^2 x_{Q'}) = y_P - \alpha\lambda' x_P + \alpha^3(\lambda' x_{Q'} - y_{Q'})$$

and $R'_3 = R' + Q'$, where $x_{R'_3} = \lambda'^2 - x_{R'} - x_{Q'}$ and $y_{R'_3} = \lambda'(x_{R'} - x_{R'_3}) - y_{R'}$. The slope λ' now is different, namely $\lambda' = (y_{R'} - y_{Q'})/(x_{R'} - x_{Q'})$. Note that $R' = -Q'$ does not occur when computing Miller function values of degree less than r. The cost for doing an addition step is the same as that for a doubling step, except that the cost to compute the slope λ' is now $\mathbf{I}_{q^e} + \mathbf{M}_{q^e} + 2\mathbf{sub}_{q^e}$.

Table 2 compares the costs for affine addition steps to those in projective coordinates. Again, we take these operation counts from the literature (see [1,16,15] for the explicit formulas and details on the computation). Concerning the field

and twist representations and line function evaluation, similar remarks as for doubling steps apply here.

The multiplication with ω in the case $d = 2$ can be done as a precomputation, since Q' is fixed throughout the pairing algorithm. Since other formulas do not have multiplications by constants, we omit this column in Table 2.

ADD	d	coord.	$\mathbf{M}_q$	$\mathbf{I}_{q^e}$	$\mathbf{M}_{q^e}$	$\mathbf{S}_{q^e}$	$\mathbf{add}_{q^e}$	$\mathbf{sub}_{q^e}$	$\mathbf{neg}_{q^e}$
$ab \neq 0$	2	affine	$k/2$	1	3	1	–	8	–
$2 \mid k$		Jacobian [1]	–	–	8	6	6	17	–
$b = 0$	4	affine	$k/4$	1	3	1	–	7	1
$4 \mid k$		$W_{(1,2)}$ [16]	$k/2$	–	9	5	7	8	1
$a = 0$	6	affine	$k/6$	1	3	1	–	7	1
$6 \mid k$		projective [15,16]	$k/3$	–	11	2	1	7	–

Table 2. Operation counts for the addition step in the ate-like Miller loop omitting $1\mathbf{M}_{q^k}$.

Affine versus projective: Doubling and addition steps for computing pairings in affine coordinates include one inversion in $\mathbb{F}_{q^e}$ per step. The various projective formulas avoid the inversion, but at the cost of doing more of the other operations. How much higher these costs are exactly, depends on the underlying field implementation and the ratio of the costs for squaring to multiplication.

A rough estimate of the counts in Table 2 shows that for $\mathbf{S}_{q^e} = \mathbf{M}_{q^e}$ or $\mathbf{S}_{q^e} = 0.8\mathbf{M}_{q^e}$ (commonly used values in the literature, see [10]), the cost traded for the inversion in the projective addition formulas is at least $9\mathbf{M}_{q^e}$. For doubling steps, it is smaller, but larger than $3\mathbf{M}_{q^e}$ in all cases. Since doubling steps are much more frequent in the pairing computation (especially when a low Hamming weight for the degree of the used Miller function is chosen), the traded cost in the doubling case is the most relevant to consider.

Example 1. Let $ab \neq 0$, i.e. $d = 2$. The cost that has to be weighed against the inversion cost for a doubling step is $9\mathbf{S}_{q^{k/2}} - (k/2)\mathbf{M}_q + \mathbf{M}_{(a/\omega^2)} - \mathbf{M}_{(\omega)} + 2\mathbf{add}_{q^{k/2}} + 11\mathbf{sub}_{q^{k/2}}$. Clearly, $(k/2)\mathbf{M}_q < \mathbf{S}_{q^{k/2}}$, and we assume $\mathbf{M}_{(\omega)} \approx \mathbf{M}_{(a/\omega^2)}$ and $\mathbf{add}_{q^{k/2}} \approx \mathbf{sub}_{q^{k/2}}$. If $\mathbf{S}_{q^{k/2}} \approx 0.8\mathbf{M}_{q^{k/2}}$, we see that if an inversion costs less than $6.4\mathbf{M}_{q^{k/2}} + 13\mathbf{add}_{q^{k/2}}$, then affine coordinates are better than Jacobian.

Example 2. In the case $a = 0$, $d = 6$, and $\mathbf{S}_{q^{k/6}} \approx 0.8\mathbf{M}_{q^{k/6}}$, similar to the previous example, we deduce that if an inversion in $\mathbb{F}_{q^{k/6}}$ is less than $3\mathbf{M}_{q^{k/6}} + (k/6)\mathbf{M}_q + \mathbf{M}_{(b/\omega)} + 12\mathbf{add}_{q^{k/6}}$, then affine coordinates beat the projective ones.

To compare affine to projective formulas, we need to look at the relative cost of an inversion that is used in the affine formulas versus the cost of the additional operations needed for the projective formulas. Therefore, an important measure that determines whether the affine formulas are competitive with the projective formulas is the ratio of the cost of an inversion to the cost of a multiplication. For a positive integer ℓ, define the inversion-to-multiplication ratio in the field $\mathbb{F}_{q^\ell}$ by $\mathbf{R}_{q^\ell} = \mathbf{I}_{q^\ell}/\mathbf{M}_{q^\ell}$.

In implementations of prime fields, inversions are usually very expensive, i.e. the ratio $\mathbf{R}_q$ is very large. So the costs for inversions are much higher than the above-mentioned costs to avoid them. Thus it does not make sense to use affine coordinates. But it is possible to obtain much smaller ratios, e.g. when computing in extension fields. Since the ate pairing requires inversions only in $\mathbb{F}_{q^e}$ this could be in favor of affine coordinates. Depending on the specific ratio $\mathbf{R}_q$ for a given implementation, affine coordinates might even be faster than projective.

3 Inversions in extension fields for the ate pairing

In this section, we describe and analyze a way to compute inversions in finite field extensions. It is based on a given, fixed implementation of arithmetic in the underlying prime field and explains that the inversion-to-multiplication ratio $\mathbf{R} = \mathbf{I}/\mathbf{M}$ decreases when moving up in a tower of field extensions.

3.1 Inverses in field extensions

The method we suggest for computing the inverse of an element in an extension of some finite field $\mathbb{F}_q$ was originally described by Itoh and Tsujii [32] for binary fields using normal bases. Kobayashi et al. [34] generalize the technique to large-characteristic fields in polynomial basis and use it for elliptic-curve arithmetic. It is a standard way to compute inverses in optimal extension fields (see [2,27] and [17, Sections 11.3.4 and 11.3.6]).

We require $\mathbb{F}_{q^\ell} = \mathbb{F}_q(\alpha)$ where α has minimal polynomial $X^\ell - \omega$ for some $\omega \in \mathbb{F}_q^*$ and assume $\gcd(\ell, q) = 1$. Then, the inverse of $\beta \in \mathbb{F}_{q^\ell}^*$ can be computed as

$$\beta^{-1} = \beta^{v-1} \cdot \beta^{-v},$$

where $v = (q^\ell - 1)/(q-1) = q^{\ell-1} + \cdots + q + 1$. Note that β^v is the norm of β and thus lies in the base field $\mathbb{F}_q$. So the cost for computing the inverse of β is the cost for computing β^{v-1} and β^v, one inversion in the base field $\mathbb{F}_q$ to obtain β^{-v}, and the multiplication of β^{v-1} with β^{-v}. The powers of β are obtained by using the q-power Frobenius automorphism on $\mathbb{F}_q^\ell$.

We give a brief estimate of the cost of the above. A Frobenius computation using a look-up table of $\ell-1$ pre-computed values in $\mathbb{F}_q$ consisting of powers of ω costs at most $\ell-1$ multiplications in $\mathbb{F}_q$ (see [34, Section 2.3], note $\gcd(\ell, q) = 1$). According to [29, Section 2.4.3] the computation of β^{v-1} via an addition chain approach, using a look-up table for each needed power of the Frobenius, costs at most $\lfloor \log(\ell-1) \rfloor + h(\ell-1)$ Frobenius computations and fewer multiplications in $\mathbb{F}_{q^\ell}$. Here $h(m)$ denotes the Hamming weight of an integer m. Knowing that $\beta^v \in \mathbb{F}_q$, its computation from β^{v-1} and β costs at most ℓ base field multiplications, one multiplication with ω, and $\ell-1$ base field additions. The final multiplication of β^{-v} with β^{v-1} can be done in ℓ base field multiplications. This leads to an upper bound for the cost of an inversion in $\mathbb{F}_{q^\ell}$ as follows:

$$\begin{aligned}\mathbf{I}_{q^\ell} \leq \mathbf{I}_q &+ (\lfloor \log(\ell-1) \rfloor + h(\ell-1))(\mathbf{M}_{q^\ell} + (\ell-1)\mathbf{M}_q) \\ &+ 2\ell\mathbf{M}_q + \mathbf{M}_{(\omega)} + (\ell-1)\mathbf{add}_q. \end{aligned} \tag{2}$$

Let $M(\ell)$ be the minimal number of multiplications in $\mathbb{F}_q$ needed to multiply two different, non-trivial elements in $\mathbb{F}_{q^\ell}$ not lying in a proper subfield of $\mathbb{F}_{q^\ell}$. Then the following lemma bounds the ratio of inversion to multiplication costs in $\mathbb{F}_{q^\ell}$ from above by $1/M(\ell)$ times the ratio in $\mathbb{F}_q$ plus an explicit constant. Thus the ratio in the extension improves by roughly a factor of $M(\ell)$.

Lemma 1. *Let $\mathbb{F}_q$ be a finite field, $\ell > 1$, $\mathbb{F}_{q^\ell} = \mathbb{F}_q(\alpha)$ with $\alpha^\ell = \omega \in \mathbb{F}_q^*$. Then using the above inversion algorithm in $\mathbb{F}_{q^\ell}$ leads to*

$$\mathbf{R}_{q^\ell} \leq \mathbf{R}_q/M(\ell) + C(\ell),$$

where $C(\ell) = \lfloor \log(\ell-1) \rfloor + h(\ell-1) + \frac{1}{M(\ell)}\big(3\ell + (\ell-1)(\lfloor \log(\ell-1) \rfloor + h(\ell-1))\big)$.

Proof. Since $M(\ell)$ is the minimal number of multiplications in $\mathbb{F}_q$ needed for multiplying two elements in $\mathbb{F}_{q^\ell}$, we can assume that the actual cost for the latter is $\mathbf{M}_{q^\ell} \geq M(\ell)\mathbf{M}_q$. Using inequality (2), we deduce

$$\mathbf{R}_{q^\ell} = \mathbf{I}_{q^\ell}/\mathbf{M}_{q^\ell} \leq \mathbf{I}_q/(M(\ell)\mathbf{M}_q) + \tilde{C}(\ell) = \mathbf{R}_q/M(\ell) + \tilde{C}(\ell),$$

where $\tilde{C}(\ell) = \lfloor \log(\ell-1) \rfloor + h(\ell-1) + (2\ell + (\ell-1)(\lfloor \log(\ell-1) \rfloor + h(\ell-1)))/M(\ell) + (\mathbf{M}_\omega + (\ell-1)\mathbf{add}_q)/(M(\ell)\mathbf{M}_q)$. Since $\mathbf{M}_{(\omega)} \leq \mathbf{M}_q$ and $\mathbf{add}_q \leq \mathbf{M}_q$, we get $\mathbf{M}_\omega + (\ell-1)\mathbf{add}_q \leq \ell\mathbf{M}_q$ and thus $\tilde{C}(\ell) \leq C(\ell)$. □

In Table 3 we give values for the factor $1/M(\ell)$ and the additive constant $C(\ell)$ that determine the improvements of $\mathbf{R}_{q^\ell}$ over $\mathbf{R}_q$ for several small extension degrees ℓ. We take the numbers for $M(\ell)$ from the formulas given in [38].

ℓ	2	3	4	5	6	7
$1/M(\ell)$	1/3	1/6	1/9	1/13	1/17	1/22
$C(\ell)$	3.33	4.17	5.33	5.08	6.24	6.05

Table 3. Constants that determine the improvement of $\mathbf{R}_{q^\ell}$ over $\mathbf{R}_q$

For small-degree extensions, the inversion method can be easily made explicit. We state and analyze it for quadratic and cubic extensions.

Quadratic extensions: Let $\mathbb{F}_{q^2} = \mathbb{F}_q(\alpha)$ with $\alpha^2 = \omega \in \mathbb{F}_q$. An element $\beta = b_0 + b_1\alpha \neq 0$ can be inverted as

$$\frac{1}{b_0 + b_1\alpha} = \frac{b_0 - b_1\alpha}{b_0^2 - b_1^2\omega} = \frac{b_0}{b_0^2 - b_1^2\omega} - \frac{b_1}{b_0^2 - b_1^2\omega}\alpha.$$

In this case the norm of β is given explicitly by $b_0^2 - b_1^2\omega \in \mathbb{F}_q$. The inverse of β thus can be computed in $\mathbf{I}_{q^2} = \mathbf{I}_q + 2\mathbf{M}_q + 2\mathbf{S}_q + \mathbf{M}_{(\omega)} + \mathbf{sub}_q + \mathbf{neg}_q$.

We assume that we multiply $\mathbb{F}_{q^2}$-elements with Karatsuba multiplication, which costs $\mathbf{M}_{q^2} = 3\mathbf{M}_q + \mathbf{M}_{(\omega)} + 2\mathbf{add}_q + 2\mathbf{sub}_q$. As in the general case above,

we assume that the cost for a full multiplication in the quadratic extension is at least $\mathbf{M}_{q^2} \geq 3\mathbf{M}_q$, i.e. we restrict to the average case where both elements have both of their coefficients different from 0. Thus

$$\mathbf{R}_{q^2} = \mathbf{I}_{q^2}/\mathbf{M}_{q^2} \leq (\mathbf{I}_q/3\mathbf{M}_q) + 2 = \mathbf{R}_q/3 + 2,$$

where we roughly assume that $\mathbf{I}_{q^2} \leq \mathbf{I}_q + 6\mathbf{M}_q$. This bound shows that for $\mathbf{R}_q > 3$ the ratio becomes smaller in $\mathbb{F}_{q^2}$. For large ratios in $\mathbb{F}_q$ it becomes roughly $\mathbf{R}_q/3$.

Cubic extensions: Let $\mathbb{F}_{q^3} = \mathbb{F}_q(\alpha)$ with $\alpha^3 = \omega \in \mathbb{F}_q$. Similar to the quadratic case we can invert $\beta = b_0 + b_1\alpha + b_2\alpha^2 \in \mathbb{F}^*_{q^3}$ by

$$\frac{1}{b_0 + b_1\alpha + b_2\alpha^2} = \frac{b_0^2 - \omega b_1 b_2}{N(\beta)} + \frac{\omega b_2^2 - b_0 b_1}{N(\beta)}\alpha + \frac{b_1^2 - b_0 b_2}{N(\beta)}\alpha^2$$

with $N(\beta) = b_0^3 + b_1^3\omega + b_2^3\omega^2 - 3\omega b_0 b_1 b_2$. We start by computing ωb_1 and ωb_2 as well as b_0^2 and b_1^2. The terms in the numerators are obtained by a 2-term Karatsuba multiplication and additions and subtractions via $3\mathbf{M}_q$ computing $b_0 b_2$, $\omega b_1 b_2$ and $(\omega b_2 + b_0)(b_2 + b_1)$. The norm can be computed by 3 more multiplications and 2 additions. Thus the cost for the inversion is $\mathbf{I}_{q^3} = \mathbf{I}_q + 9\mathbf{M}_q + 2\mathbf{S}_q + 2\mathbf{M}_{(\omega)} + 4\mathbf{add}_q + 4\mathbf{sub}_q$. A Karatsuba multiplication can be done in $\mathbf{M}_{q^3} = 6\mathbf{M}_q + 2\mathbf{M}_{(\omega)} + 9\mathbf{add}_q + 6\mathbf{sub}_q$. We use $\mathbf{M}_{q^3} \geq 6\mathbf{M}_q$, assume $\mathbf{I}_{q^3} \leq \mathbf{I}_q + 18\mathbf{M}_q$ and obtain $\mathbf{R}_{q^3} = \mathbf{I}_{q^3}/\mathbf{M}_{q^3} \leq (\mathbf{I}_q/6\mathbf{M}_q) + 3 = \mathbf{R}_q/6 + 3$.

Towers of field extensions: Baktir and Sunar [3] introduce optimal tower fields as an alternative for optimal extension fields, where they build a large field extension as a tower of small extensions instead of one big extension. They describe how to use the above inversion technique recursively by passing down the inversion in the tower, finally arriving at the base field. They show that this method is more efficient than computing the inversion in the corresponding large extension with the Itoh-Tsujii inversion directly.

In pairing-based cryptography it is common to use towers of fields to represent the extension $\mathbb{F}_{q^k}$, where k is the embedding degree. Benger and Scott [9] discuss how to best choose such towers, but do not address inversions.

3.2 Extension-field inversions for the ate pairing

We have seen in Section 2 that for the ate pairing, the inversions in the doubling and addition steps are inversions in a proper extension field of $\mathbb{F}_q$. We now take a closer look at specific high-security levels to see which degrees these extension fields have. For a pairing-friendly elliptic curve E over $\mathbb{F}_q$ with embedding degree k with respect to a prime divisor $r \mid \#E(\mathbb{F}_q)$, we define the ρ-value of E as $\rho = \log(q)/\log(r)$. This value is a measure of the base field size relative to the size of the prime-order subgroup on the curve.

Table 4 gives the recommendations by NIST [4] and ECRYPT II [43] for equivalent levels of security for the discrete logarithm problems in the elliptic

curve subgroup of order r and in a subgroup of $\mathbb{F}_{q^k}^*$. For efficiency reasons, it is desirable to balance the security in both groups. The group sizes are linked by the embedding degree k, which leads to desired values for ρk as given in Table 4.

security		NIST [4]		ECRYPT II [43]	
(bits)	r (bits)	q^k (bits)	ρk	q^k (bits)	ρk
128	256	3072	12	3248	12.69
192	384	7680	20	7936	20.67
256	512	15360	30	15424	30.13

Table 4. NIST [4] and ECRYPT II [43] recommendations for bitsizes of r and q^k providing equivalent levels of security on elliptic-curve point groups and in finite fields.

To implement pairings at a given security level, one needs to find a pairing-friendly elliptic curve with parameters of at least the sizes given in Table 4; for efficiency it is even desirable to obtain ρk as close to the desired value as possible. An overview of construction methods for pairing-friendly elliptic curves is given in [20]. In Table 5, we list suggestions for curve families by their construction in [20] for high-security levels of 128, 192, and 256 bits. The last column in Table 5 shows the field extensions in which inversions are done to compute the line function slopes. We not only give families of curves with twists of degree 4 and 6, but also more generic families such that the curves only have a twist of degree 2. Of course, in the latter case the extension field, in which inversions for the affine ate pairing need to be computed, is larger than when dealing with higher-degree twists. Because curves with twists of degree 4 and 6 are special (they have j-invariants 1728 and 0), there might be reasons to choose the more generic curves. Note that curves from the given constructions are all defined over prime fields. Therefore we use the notation $\mathbb{F}_p$ in Table 5.

security	construction in [20]	curve	k	ρ	ρk	d	extension
128	Ex. 6.8	$a = 0$	12	1.00	12.00	6	$\mathbb{F}_{p^2}$
	Ex. 6.10	$b = 0$	8	1.50	12.00	4	$\mathbb{F}_{p^2}$
	Section 5.3	$a, b \neq 0$	10	1.00	10.00	2	$\mathbb{F}_{p^5}$
	Constr. 6.7+	$a, b \neq 0$	12	1.75	21.00	2	$\mathbb{F}_{p^6}$
192	Ex. 6.12	$a = 0$	18	1.33	24.00	6	$\mathbb{F}_{p^3}$
	Ex. 6.11	$b = 0$	16	1.25	20.00	4	$\mathbb{F}_{p^4}$
	Constr. 6.3+	$a, b \neq 0$	14	1.50	21.00	2	$\mathbb{F}_{p^7}$
256	Constr. 6.6	$a = 0$	24	1.25	30.00	6	$\mathbb{F}_{p^4}$
	Constr. 6.4	$b = 0$	28	1.33	37.24	4	$\mathbb{F}_{p^7}$
	Constr. 6.24+	$a, b \neq 0$	26	1.17	30.34	2	$\mathbb{F}_{p^{13}}$

Table 5. Extension fields for which inversions are needed when computing ate-like pairings for different examples of pairing-friendly curve families suitable for the given security levels.

Remark 1. The conclusion to underline from the discussion in this section, is that, using the improved inversions in towers of extension fields described here, there are at least two scenarios where *most implementations of the ate pairing would be more efficient using affine coordinates*:

When higher security levels are required, so that k is large. For example 256-bit security with $k = 28$, so that most of the computations for the ate pairing take place in the field extension of degree 7, even using a degree-4 twist (second-to-last line of Table 5). In that case, the $\mathbf{I}/\mathbf{M}$ ratio in the degree-7 extension field would be roughly 22 times less (plus 6) than the ratio in the base field (see the last entry in Table 3). The costs for doubling and addition steps given in the second lines of Tables 1 and 2 for degree-4 twists show that the cost of the inversion avoided in a projective implementation should be compared with roughly $6\mathbf{S}_{q^7} + 5\mathbf{add}_{q^7} + 5\mathbf{sub}_{q^7}$ extra for a doubling (and an extra $6\mathbf{M}_{q^7} + 4\mathbf{S}_{q^7} + 7\mathbf{add}_{q^7} + \mathbf{sub}_{q^7}$ for an addition step). In most implementations of the base field arithmetic, the cost of these 16 or 17 operations in the extension field would outweigh the cost of one improved inversion in the extension field. See for example our sample timings for degree-6 extension fields in Table 6 in Section 5. Note there that even the cost for additions and subtractions is not negligible as is usually assumed.

When special high-degree twists are not being used. In this scenario there are two reasons why affine coordinates will be better under most circumstances:

First, the costs for doubling and addition steps given in the first lines of Tables 1 and 2 for degree-2 twists are not nearly as favorable towards projective coordinates as the formulas in the case of higher degree twists. For degree-2 twists, both the doubling and addition steps require roughly at least 9 extra squarings and 13 or 15 extra field extension additions or subtractions for the projective formulas.

Second, the degree of the extension field where the operations take place is larger. See the bottom row for each security level in Table 5, so we have extension degree 6 for 128-bit security up to extension degree 13 for 256-bit security.

4 Sharing inversions for pairing computation

In this section, we revisit a well-known trick for efficiently computing several inverses at once, asymptotically achieving an $\mathbf{I}/\mathbf{M}$-ratio of 3. We point out and recall possibilities to improve pairing computation in affine coordinates by using this trick.

4.1 Simultaneous inversions

The inverses of s field elements $a_1, \ldots, a_s$ can be computed simultaneously with Montgomery's well-known sharing-inversions trick [37, Section 10.3.1.] at the cost of 1 inversion and $3(s-1)$ multiplications. It is based on the following idea: to compute the inverse of two elements a and b, one computes their product ab

and its inverse $(ab)^{-1}$. The inverses of a and b are then found by $a^{-1} = b \cdot (ab)^{-1}$ and $b^{-1} = a \cdot (ab)^{-1}$.

In general, for s elements one first computes the products $c_i = a_1 \cdot \cdots \cdot a_i$ for $2 \leq i \leq s$ with $s-1$ multiplications and inverts c_s. Then we have $a_s^{-1} = c_{s-1}c_s^{-1}$. We get a_{s-1}^{-1} by $c_{s-1}^{-1} = c_s^{-1}a_s$ and $a_{s-1}^{-1} = c_{s-2}c_{s-1}^{-1}$ and so forth (see [17, Algorithm 11.15]), where we need $2(s-1)$ more multiplications to get the inverses of all elements.

The cost for s inversions is replaced by $\mathbf{I} + 3(s-1)\mathbf{M}$. Let $\mathbf{R}_{\text{avg},s}$ denote the ratio of the cost of s inversions to the cost of s multiplications. It is bounded above by $\mathbf{R}_{\text{avg},s} = \mathbf{I}/(s\mathbf{M}) + 3(s-1)/s \leq \mathbf{R}/s + 3$, i.e. when the number s of elements to be inverted grows, the ratio $\mathbf{R}_{\text{avg},s}$ gets closer to 3. Note that most of the time, this method improves the efficiency of an implementation whenever applicable. However, as discussed in Section 3, in large field extensions, the $\mathbf{I}/\mathbf{M}$-ratio might already be less than 3 due to the inversion method from Section 3.1, in which case the sharing trick would make the average ratio worse.

4.2 Sharing inversions in a single pairing computation

Schroeppel and Beaver [40] demonstrate the use of the inversion-sharing trick to speed up a single scalar multiplication on an elliptic curve in affine coordinates. They suggest postponing addition steps in the double-and-add algorithm to exploit the inversion sharing. In order to do that, the double-and-add algorithm must be carried out by going through the binary representation of the scalar from right to left. First, all doublings are carried out and the points that will be used to add up to the final result are stored. When all these points have been collected, several additions can be done at once, sharing the computation of inversions among them.

Miller's algorithm can also be done from right to left. The doubling steps are computed without doing the addition steps. The required field elements and points are stored in lists and addition steps are done in the end. The algorithm is summarized in Algorithm 2. Unfortunately, addition steps cost much more than in the conventional left-to-right algorithm as it is given in Algorithm 1. In the right-to-left version, each addition step in Line 10 needs a general $\mathbb{F}_{q^k}$-multiplication and a multiplication with a line function value. The conventional algorithm only needs a multiplication with a line. These huge costs can not be compensated by using affine coordinates with the inversion-sharing trick.

Parallelizing a single pairing. However, the right-to-left algorithm can be parallelized, and this could lead to more efficient implementations taking advantage of the recent advent of many-core machines. Grabher, Großschädl, and Page [23, Algorithm 2] use a version of Algorithm 2 to compute a single pairing by doing addition steps in parallel on two different cores. They divide the lists with the saved function values and points into two halves and compute two intermediate values which are in the end combined in a single addition step. For their specific implementation, they conclude that this is not faster than the conventional non-parallel algorithm. Still, this idea might be useful for two or more

Algorithm 2 Right-to-left version of Miller's algorithm with postponed addition steps for even k and ate-like pairings

Input: $Q' \in G_2', P \in G_1, m = (1 = m_{l-1}, m_{l-2}, \ldots, m_0)_2$
Output: $f_{m,\psi(Q')}(P)$ representing a class in $\mathbb{F}_{q^k}^*/(\mathbb{F}_{q^k}^*)^r$

1: $R' \leftarrow Q', \quad f \leftarrow 1, \quad j \leftarrow 0$
2: **for** i from 0 to $\ell - 1$ **do**
3: **if** $(m_i = 1)$ **then**
4: $A_{R'}[j] \leftarrow R', \quad A_f[j] \leftarrow f, \quad j \leftarrow j + 1$
5: **end if**
6: $f \leftarrow f^2 \cdot l_{\psi(R'),\psi(R')}(P), \quad R' \leftarrow [2]R'$
7: **end for**
8: $R' \leftarrow A_{R'}[0], \quad f \leftarrow A_f[0]$
9: **for** $(j \leftarrow 1;\ j \leq h(m) - 1; j++)$ **do**
10: $f \leftarrow f \cdot A_f[j] \cdot l_{\psi(R'),\psi(A_{R'}[j])}(P), \quad R' \leftarrow R' + A_{R'}[j]$
11: **end for**
12: **return** f

cores, once multiple cores can be used with less overhead. It is straightforward to extend this algorithm to more cores.

So we suggest that the parallelized algorithm can be combined with the shared inversion trick when doing the addition steps in the end. The improvements achieved by this approach strongly depend on the Hamming weight of the value m in Miller's algorithm. If it is large, then savings are large, while for very sparse m there is almost no improvement. Therefore, when it is not possible to choose m with low Hamming weight, combining the parallelized right-to-left algorithm for pairings with the shared inversion trick can speed-up the computation. Grabher et al. [23] note that when multiple pairings are computed, it is better to parallelize by performing one pairing on each core.

4.3 Multiple pairings and products of pairings

Many protocols involve the computation of multiple pairings or products of pairings. For example, multiple pairings need to be computed in the searchable encryption scheme of Boneh et al. [13]; and the non-interactive proof systems proposed by Groth and Sahai [26] need to check pairing product equations. In these scenarios, we propose sharing inversions when computing pairings with affine coordinates. In the case of products of pairings, this has already been proposed and investigated by Scott [41, Section 4.3] and Granger and Smart [25].

Multiple pairings. Assume we want to compute s pairings on points Q_i' and P_i, i.e. a priori we have s Miller loops to compute $f_{m,\psi(Q_i')}(P_i)$. We carry out these loops simultaneously, doing all steps up to the first inversion computation for a line function slope for all of them. Only after that, all slope denominators are inverted simultaneously, and we continue with the computation for all pairings until the next inversion occurs. The s Miller loops are not computed sequentially,

but rather sliced at the slope denominator inversions. The costs stay the same, except that now the average inversion-to-multiplication cost ratio is $3 + \mathbf{R}_{q^e}/s$, where $e = k/d$ and d is the twist degree.

So when computing enough pairings such that the average cost of an inversion is small enough, using the sliced-Miller approach with inversion sharing in affine coordinates is faster than using the projective coordinates explicit formulas described in Section 2.2.

Products of pairings. For computing a product of pairings, more optimizations can be applied, including the above inversion-sharing. Scott [41, Section 4.3] suggests using affine coordinates and sharing the inversions for computing the line function slopes as described above for multiple pairings. Furthermore, since the Miller function of the pairing product is the product of the Miller functions of the single pairings, in each doubling and addition step the line functions can already be multiplied together. In this way, we only need one intermediate variable f and only one squaring per iteration of the product Miller loop. Of course in the end, there is only one final exponentiation on the product of the Miller function values. Granger and Smart [25] show that by using these optimizations the cost for introducing an additional ate pairing to the product can be as low as 13% of the cost of a single ate pairing.

5 Example implementation

The implementation described in this section is an implementation of the optimal ate pairing on a Barreto-Naehrig (BN) curve [8] over a 256-bit prime field, i.e. the curve has a 256-bit prime number n of $\mathbb{F}_p$-rational points and embedding degree $k = 12$ with respect to n.

The implementation is part of the Microsoft Research pairing library. It is specialized to the BN curve family but is not specialized for a specific BN curve. It is based on Microsoft Research's general purpose library for big number arithmetic, which can be compiled under 32-bit or 64-bit Windows. On top of that, we use the tower of field extensions $\mathbb{F}_{p^{12}}/\mathbb{F}_{p^6}/\mathbb{F}_{p^2}/\mathbb{F}_p$ to realize field arithmetic in $\mathbb{F}_{p^{12}}$. In Table 6 we give timings for the required field arithmetic in the fields $\mathbb{F}_p$, $\mathbb{F}_{p^2}$, $\mathbb{F}_{p^6}$, and $\mathbb{F}_{p^{12}}$ for the 32-bit and 64-bit versions, respectively. The 32-bit timings are for a pure software C-implementation, while the 64-bit software makes use of assembly code for base field multiplications, i.e. special code for Montgomery multiplication with a prime modulus of 256 bits, only using the fixed size of the modulus. Note that the timings in cycles and miliseconds stem from two different measurements and thus do not exactly translate.

The last column in Table 6 gives the $\mathbf{I}/\mathbf{M}$-ratios for the corresponding extension field and demonstrates the effect of using the inversion method for extension field towers described in Section 3.1. The ratios are even smaller than predicted by the theoretical upper bounds in Lemma 1 and Table 3. This is explained by the fact that actual multiplication costs for elements in $\mathbb{F}_{q^\ell}$ are higher than the

estimates given there that take into account only multiplications from the base field and neglect all other base field operations.

		add		**sub**		**M**		**S**		**I**		**R = I/M**
		cyc	μs	cyc	μs	cyc	μs	cyc	μs	cyc	μs	
32-bit	$\mathbb{F}_p$	327	0.11	309	0.10	988	0.32	945	0.32	13285	4.18	13.45
	$\mathbb{F}_{p^2}$	588	0.19	585	0.18	4531	1.44	2949	0.91	18687	5.65	4.13
	$\mathbb{F}_{p^6}$	1746	0.54	1641	0.52	38938	12.09	26364	8.44	78847	24.98	2.03
	$\mathbb{F}_{p^{12}}$	3300	1.06	3233	1.03	123386	38.97	88249	27.94	210907	66.90	1.71
64-bit	$\mathbb{F}_p$	189	0.06	163	0.05	414	0.13	414	0.13	9469	2.98	22.87
	$\mathbb{F}_{p^2}$	329	0.10	300	0.10	2122	0.67	1328	0.42	11426	3.65	5.38
	$\mathbb{F}_{p^6}$	931	0.29	834	0.26	18544	5.81	12929	4.05	40201	12.66	2.17
	$\mathbb{F}_{p^{12}}$	1855	0.57	1673	0.51	60967	19.17	43081	13.57	103659	32.88	1.70

Table 6. Field arithmetic timings in a 256-bit prime field, on an Intel Core 2 Duo E8500 @ 3.16 GHz under 32-bit/64-bit Windows 7. Average over 1000 operations in cpucycles (cyc) and microseconds (μs).

The pairing implementation uses the usual optimizations. First of all, a twist $E'/\mathbb{F}_{p^2}$ provides the group G_2' to represent elements in G_2 as described in Section 2.1. The affine doubling and addition steps in Miller's algorithm are computed as shown in Section 2.2. The projective steps use the explicit formulas from the recent paper of Costello et al. [16]. The final exponentiation is done as described in [42], and uses the special squaring formulas given by Granger and Scott [24].

Table 7 gives benchmarking results for several pairing functions in the library, compiled under 32-bit and 64-bit Windows 7, respectively. All functions compute the optimal ate pairing for BN curves as described for example in [39]. The line entitled "20 at once (per pairing)" gives the average timing for one pairing out of 20 that have been computed at the same time. This function uses the inversion-sharing trick as described in Section 4.3. The function corresponding to the line "product of 20" computes the product of 20 pairings using the optimizations described in Section 4.3. The lines with the attribute "1st arg. fixed" mean functions that compute multiple pairings or a product of pairings, where the first input point is fixed for all pairings, and only the second point varies. In this case, the operations depending only on the first argument are done only once. We list separately the final exponentiation timings. They are included in the pairing timings of the other lines.

Implementation notes.

1. For both the 32-bit and 64-bit versions of the library, a single pairing is computed faster with affine coordinates than with projective coordinates. This is due to the relatively low $\mathbf{I}/\mathbf{M}$-ratios in the base field $\mathbb{F}_p$ (13.45 and 22.87 respectively) and in the quadratic extension (ratios 4.13 and 5.38 respectively). These low ratios are due to a relatively efficient inversion implementation in

optimal ate pairings		32-bit		64-bit	
		cyc	ms	cyc	ms
projective		32,288,630	10.06	15,426,503	4.88
affine	single pairing	30,091,044	9.49	14,837,947	4.64
	20 at once (per pairing)	29,681,288	9.39	14,442,433	4.53
	20 at once, 1st arg. fixed (per pairing)	27,084,852	8.53	13,124,802	4.12
	product of 20 (per pairing)	10,029,724	3.16	4,832,725	1.52
	product of 20, 1st arg. fixed (per pairing)	7,316,501	2.32	3,563,108	1.12
single final exponentiation		15,043,435	4.75	7,266,020	2.28

Table 7. Optimal ate pairing timings on a 256-bit BN curve, measured on an Intel Core 2 Duo E8500 @ 3.16 GHz under 32-bit/64-bit Windows 7. Average over 20 pairings in cpucycles (cyc) and milliseconds (ms).

the base field combined with the improved inversion for quadratic extensions given in Section 3.1.

2. At this security level (128-bits) and using the special high-degree-6 twist, the projective implementation is almost on par with the affine implementation, so that even a small improvement in the base field multiplication would tip the balance in favor of a projective implementation.
3. However, as was explained in Remark 1 in Section 3.2, either for higher security levels or for curves without special high degree twists, affine coordinates will be much faster than projective coordinates given our base field and extension field arithmetic. Indeed, our **I**/**M**-ratio in a degree 6 extension is already roughly 2, for both our 32-bit and 64-bit versions. With a ratio of 2, projective coordinates are not a good choice.
4. Because our **I**/**M**-ratios in the quadratic field extension are already so close to 3, there is little improvement expected or observed from using the shared inversion tricks discussed in Section 4.
5. Note that field addition and subtraction costs are not negligible, as one might think from the fact that they are not often included in the operation counts when comparing various methods for elliptic curve operations and pairing implementations. In our base field arithmetic, 1 multiplication costs roughly the same as 3 field additions or subtractions, but the relative cost of additions and subtractions in extension fields is significantly less.
6. Note that the ratio of squarings to multiplications changes in the extension fields as well. A squaring in the quadratic extension is done with only 2 multiplications using the fact that the extension is generated by $\sqrt{-1}$. This improvement carries through to squarings in the higher field extensions.

Comparison to related work. We compare our work with the best results for optimal ate pairing implementations on BN curves that we are aware of.

The software described in [28] needs about $10,000,000$ cycles on an Intel Core 2 for the R-ate pairing. Modular multiplication takes 310 cycles which is about 25% faster than ours and seems to mostly account for the difference in performance with our implementation for a pairing in projective coordinates.

Recently, there has been significant improvement on pairing implementations for BN curves. The paper [39] presents an implementation that computes the optimal ate pairing on a 256-bit BN curve using one core of an Intel Core 2 Quad in about $4,380,000$ cycles. The implementation described in [11] computes the same pairing on a 254-bit BN curve in $2,490,000$ cycles on an Intel Core i7.

Software as described in [39] and [11] is much faster than our implementation for the following reason. The above implementations gain their efficiency by special curve parameter choices combined with a careful instruction scheduling specific to the parameters and certain computer architectures or even processors, in particular resulting in highly efficient multiplications in the base field and the quadratic extension field. Instead, our implementation is based on a general-purpose library for the base field arithmetic which can be compiled on many platforms and works for all BN curves. Thus our implementation is not competitive with specially tailored ones as in [39] and [11]. Nevertheless, the effects implied by the use of affine coordinates that we demonstrated with the help of our implementation also apply to implementations with faster field multiplications. Affine coordinates will then be better only when working with larger extension degrees that occur for higher security levels.

Acknowledgements: We would like to thank Dan Shumow and Tolga Acar for their help with the development environment for our implementation. We thank Steven Galbraith, Diego F. Aranha, and the anonymous referees for their helpful comments to improve the paper.

References

1. Christophe Arène, Tanja Lange, Michael Naehrig, and Christophe Ritzenthaler. Faster computation of the Tate pairing. *Journal of Number Theory*, 2010. doi:10.1016/j.jnt.2010.05.013.
2. Daniel V. Bailey and Christof Paar. Efficient arithmetic in finite field extensions with application in elliptic curve cryptography. *Journal of Cryptology*, 14(3):153–176, 2001.
3. Selçuk Baktir and Berk Sunar. Optimal tower fields. *IEEE Transactions on Computers*, 53(10):1231–1243, 2004.
4. Elaine Barker, William Barker, William Burr, William Polk, and Miles Smid. Recommendation for key management - part 1: General (revised). Technical report, NIST National Institute of Standards and Technology, 2007. Published as NIST Special Publication 800-57, http://csrc.nist.gov/groups/ST/toolkit/documents/SP800-57Part1_3-8-07.pdf.
5. Paulo S. L. M. Barreto, Steven D. Galbraith, Colm Ó hÉigeartaigh, and Michael Scott. Efficient pairing computation on supersingular abelian varieties. *Designs, Codes and Cryptography*, 42(3):239–271, 2007.
6. Paulo S. L. M. Barreto, Hae Yong Kim, Ben Lynn, and Michael Scott. Efficient algorithms for pairing-based cryptosystems. In *CRYPTO 2002*, volume 2442 of *Lecture Notes in Computer Science*, pages 354–368. Springer, 2002.
7. Paulo S. L. M. Barreto, Ben Lynn, and Michael Scott. Efficient implementation of pairing-based cryptosystems. *Journal of Cryptology*, 17(4):321–334, 2004.

8. Paulo S. L. M. Barreto and Michael Naehrig. Pairing-friendly elliptic curves of prime order. In *Selected Areas in Cryptography – SAC 2005*, volume 3897 of *Lecture Notes in Computer Science*, pages 319–331. Springer, 2006.
9. Naomi Benger and Michael Scott. Constructing tower extensions of finite fields for implementation of pairing-based cryptography. In M. Anwar Hasan and Tor Helleseth, editors, *Arithmetic of Finite Fields – WAIFI 2010, Istanbul, Turkey*, volume 6087 of *Lecture Notes in Computer Science*, pages 180–195. Springer, 2010.
10. Daniel J. Bernstein and Tanja Lange. Explicit-formulas database. `http://www.hyperelliptic.org/EFD`.
11. J.-L. Beuchat, J. E. González Díaz, S. Mitsunari, E. Okamoto, F. Rodríguez-Henríquez, and T. Teruya. High-speed software implementation of the optimal ate pairing over Barreto-Naehrig curves. IACR ePrint Archive, report 2010/354, 2010. `http://eprint.iacr.org/2010/354`.
12. Ian F. Blake, Gadiel Seroussi, and Nigel P. Smart, editors. *Advances in Elliptic Curve Cryptography*. Cambridge University Press, 2005.
13. Dan Boneh, Giovanni Di Crescenzo, Rafail Ostrovsky, and Giuseppe Persiano. Public key encryption with keyword search. In *EUROCRYPT 2004*, volume 3027 of *Lecture Notes in Computer Science*, pages 506–522. Springer, 2004.
14. Henri Cohen, Gerhard Frey, and Christophe Doche, editors. *Handbook of Elliptic and Hyperelliptic Curve Cryptography*. Chapman and Hall/CRC, 2005.
15. Craig Costello, Hüseyin Hisil, Colin Boyd, Juan Manuel González Nieto, and Kenneth Koon-Ho Wong. Faster pairings on special Weierstrass curves. In *Pairing 2009*, volume 5671 of *Lecture Notes in Computer Science*, pages 89–101. Springer, 2009.
16. Craig Costello, Tanja Lange, and Michael Naehrig. Faster pairing computations on curves with high-degree twists. In *Public-Key Cryptography – PKC 2010*, volume 6056 of *Lecture Notes in Computer Science*, pages 224–242. Springer, 2010.
17. Christophe Doche. *Finite Field Arithmetic*, chapter 11 in [14], pages 201–237. CRC press, 2005.
18. Sylvain Duquesne and Gerhard Frey. *Background on Pairings*, chapter 6 in [14], pages 115–124. CRC press, 2005.
19. Sylvain Duquesne and Gerhard Frey. *Implementation of Pairings*, chapter 16 in [14], pages 389–404. CRC press, 2005.
20. David Freeman, Michael Scott, and Edlyn Teske. A taxonomy of pairing-friendly elliptic curves. *Journal of Cryptology*, 23(2):224–280, 2010.
21. Steven D. Galbraith. *Pairings*, chapter IX in [12], pages 183–213. Cambridge University Press, 2005.
22. Steven D. Galbraith, Keith Harrison, and David Soldera. Implementing the Tate pairing. In Claus Fieker and David R. Kohel, editors, *ANTS-V*, volume 2369 of *Lecture Notes in Computer Science*, pages 324–337. Springer, 2002.
23. Philipp Grabher, Johann Großschädl, and Dan Page. On software parallel implementation of cryptographic pairings. In *SAC 2008*, volume 5381 of *Lecture Notes in Computer Science*, pages 35–50. Springer, 2009.
24. Robert Granger and Michael Scott. Faster squaring in the cyclotomic group of sixth degree extensions. In *Public-Key Cryptography – PKC 2010*, volume 6056 of *Lecture Notes in Computer Science*, pages 209–223. Springer, 2010.
25. Robert Granger and Nigel P. Smart. On computing products of pairings. Cryptology ePrint Archive, Report 2006/172, 2006. `http://eprint.iacr.org/2006/172/`.
26. Jens Groth and Amit Sahai. Efficient non-interactive proof systems for bilinear groups. In *EUROCRYPT 2008*, volume 4965 of *Lecture Notes in Computer Science*, pages 415–432. Springer, 2008.

27. Jorge Guajardo and Christof Paar. Itoh-Tsujii inversion in standard basis and its application in cryptography and codes. *Designs, Codes and Cryptography*, 25:207–216, 2001.
28. Darrel Hankerson, Alfred J. Menezes, and Michael Scott. Software implementation of pairings. In Marc Joye and Gregory Neven, editors, *Identity-Based Cryptography - Volume 2 Cryptology and Information Security Series*. IOS Press, Amsterdam, The Netherlands, The Netherlands, 2008.
29. Darrel Hankerson, Alfred J. Menezes, and Scott Vanstone. *Guide to Elliptic Curve Cryptography*. Springer New York, Inc., Secaucus, NJ, USA, 2003.
30. Florian Heß, Nigel P. Smart, and Frederik Vercauteren. The eta pairing revisited. *IEEE Transactions on Information Theory*, 52:4595–4602, 2006.
31. Sorina Ionica and Antoine Joux. Another approach to pairing computation in Edwards coordinates. In *Progress in Cryptology – INDOCRYPT 2008*, volume 5365 of *Lecture Notes in Computer Science*, pages 400–413. Springer, 2008.
32. Toshiya Itoh and Shigeo Tsujii. A fast algorithm for computing multiplicative inverses in GF(2ˆm) using normal bases. *Inf. Comput.*, 78(3):171–177, 1988.
33. Tetsuya Izu and Tsuyoshi Takagi. Efficient computations of the Tate pairing for the large MOV degrees. In *Information Security and Cryptology - ICISC 2002, Seoul, Korea, November 28-29, 2002, Revised Papers*, volume 2587 of *Lecture Notes in Computer Science*, pages 283–297. Springer-Verlag, 2003.
34. Tetsutaro Kobayashi, Hikaru Morita, Kunio Kobayashi, and Fumitaka Hoshino. Fast elliptic curve algorithm combining Frobenius map and table reference to adapt to higher characteristic. In *Advances in Cryptology - EUROCRYPT 1999*, volume 1592 of *Lecture Notes in Computer Science*, pages 176–189. Springer, 1999.
35. E. Lee, H. S. Lee, and C.-M. Park. Efficient and generalized pairing computation on Abelian varieties. *IEEE Trans. on Information Theory*, 55(4):1793–1803, 2009.
36. Victor S. Miller. The Weil pairing and its efficient calculation. *Journal of Cryptology*, 17(4):235–261, 2004.
37. Peter L. Montgomery. Speeding the Pollard and elliptic curve methods of factorization. *Mathematics of Computation*, 48(177):243–264, 1987.
38. Peter L. Montgomery. Five, six, and seven-term Karatsuba-like formulae. *IEEE Transactions on Computers*, 54(3):362–369, 2005.
39. Michael Naehrig, Ruben Niederhagen, and Peter Schwabe. New software speed records for cryptographic pairings. In *Progress in Cryptology – Latincrypt 2010*, volume 6212 of *Lecture Notes in Computer Science*, pages 109–123. Springer, 2010. Corrected version: `http://www.cryptojedi.org/papers/dclxvi-20100714.pdf`.
40. Rich Schroeppel and Cheryl Beaver. Accelerating elliptic curve calculations with the reciprocal sharing trick. Mathematics of Public-Key Cryptography (MPKC), University of Illinois at Chicago, 2003.
41. Michael Scott. Computing the Tate pairing. In *CT-RSA*, volume 3376 of *Lecture Notes in Computer Science*, pages 293–304. Springer, 2005.
42. Michael Scott, Naomi Benger, Manuel Charlemagne, Luis J. Dominguez Perez, and Ezekiel J. Kachisa. On the final exponentiation for calculating pairings on ordinary elliptic curves. In *Pairing-Based Cryptography – Pairing 2009*, volume 5671 of *Lecture Notes in Computer Science*, pages 78–88. Springer, 2009.
43. Nigel Smart (editor). ECRYPT II yearly report on algorithms and keysizes (2009-2010). Technical report, ECRYPT II – European Network of Excellence in Cryptology, EU FP7, ICT-2007-216676, 2010. Published as deliverable D.SPA.13, `http://www.ecrypt.eu.org/documents/D.SPA.13.pdf`.
44. Frederik Vercauteren. Optimal pairings. *IEEE Transactions on Information Theory*, 56(1):455–461, 2010.