Cryptographic pairings

Kristin Lauter and Michael Naehrig

Microsoft Research, Redmond, USA

Abstract

This article appeared as Chapter 9 of the book *Topics in Computational Number Theory inspired by Peter L. Montgomery*, edited by Joppe W. Bos and Arjen K. Lenstra and published by Cambridge University Press. See `www.cambridge.org/9781107109353`. There are cross-references to four chapters of the same book: Chapter 1 *Introduction* by Joppe W. Bos, Arjen K. Lenstra, Herman te Riele, and Daniel Shumow, Chapter 2 *Montgomery arithmetic from a software perspective* by Joppe W. Bos, Chapter 4 *Montgomery curves and the Montgomery ladder* by Daniel J. Bernstein and Tanja Lange, and Chapter 8 *FFT extension for algebraic-group factorization algorithms* by Richard P. Brent, Alexander Kruppa, and Paul Zimmermann.

Contents

9

Cryptographic Pairings

Kristin Lauter and Michael Naehrig

During more than 15 years as a Principal Researcher at Microsoft Research, Peter L. Montgomery contributed substantially to building the public key cryptography libraries for Microsoft. The field of Elliptic Curve Cryptography (ECC) was scarcely 15 years old, and pairing-based cryptography had been recently introduced, when Peter started working on implementing and optimizing cryptographic pairings on elliptic and hyperelliptic curves.

By *pairings* on elliptic curves in the cryptographic setting, we are referring to bilinear maps from the group of points on an elliptic curve to the multiplicative group of a finite field, most notably the *Weil* pairing. Cryptographic pairings became a hot topic after the introduction of solutions for various interesting cryptographic primitives, including identity-based non-interactive key agreement [55], one-round tripartite Diffie–Hellman key exchange [44, 45], identity-based encryption [16] and short signatures [18, 19]. A flood of other cryptographic applications and constructions followed, such as attribute-based encryption [54], functional encryption [20], and homomorphic encryption [17], to name a few. Pairings originally played a key role in the earlier Menezes-Okamoto-Vanstone [49] and Frey-Rück [31] attacks on supersingular elliptic curves, which had instead negative implications for cryptographic primitives using such curves. All cryptographic applications of pairings rely on the ability to find suitable elliptic curve parameters and the existence of efficient algorithms to compute in the groups involved in the pairing operation and for the pairing computation itself.

This chapter is based on the three papers, [27], [28] and [48] on pairing computation, which Peter coauthored with one or both of the authors. The text contains excerpts from those works, lightly edited with adjustments made to unify notation and embed into the context of this chapter.

One of Peter's guiding principles was to always write and prove everything in the most general possible case, which led him, for example, to insist on

giving formulas for the often pesky case of characteristic two. The code he wrote was usually intended to be most general in the sense that it should allow to be compiled and run efficiently and securely on a wide variety of possible processors. And he produced side-channel resistant implementations without even mentioning the existence of such attacks. In his implementations of elliptic curve cryptography, Peter often used Montgomery multiplication for the base field arithmetic, assuming operations would be carried out modulo general primes, and would not rely on generalized Mersenne primes or other primes with special reduction, for example. Peter used to significantly speed up modular inversion in his implementations, which led to a low ratio for the cost of inversion to multiplication.

Several other signature ideas from Peter's work played a role in optimizing algorithms for pairing computation. In addition to fast modular inversion, Peter also introduced the widely used simultaneous inversion trick [52, Section 10.3.1.] (see Chapter 1), which can be applied to pairing computation as explained in Section 9.3 on page 14. In general, achieving a low ratio of inversion to multiplication cost through this collection of tricks in various settings favors affine coordinates, which is relevant, for example, in application scenarios such as the computation of multiple pairings, products of pairings, and parallelized pairings, as well as for pairings at very high security levels. These applications are all explained in Section 9.3.

In another direction, the Montgomery ladder [52] (see Chapter 4) used in the ECM factoring method (see Chapter 8) does not need the y-coordinate of the elliptic curve points and allows the efficient computation of scalar multiplication using x-coordinates only. In Section 9.4 on page 22, we present the double-and-add trick from [27], which saves a multiplication in combined operations by not computing the intermediate y-coordinate. Also, the squared Weil and Tate pairings, presented in Section 9.5 on page 26, achieve cancellation of vertical line function contributions because they only depend on the x-coordinates, which are equal for a point and its negative.

A common theme throughout this chapter is the comparison of affine versus projective coordinates in the elliptic curve operations involved during a pairing computation. While early papers on pairing implementation naturally started out using affine coordinates [8, 33], it is now understood that for a single pairing computation at the 128-bit and 192-bit security levels, projective coordinates are advantageous, see the results in [12, 3, 1, 2].

9.1 Preliminaries

We start by introducing notation and describing the basic concepts needed to talk about cryptographic pairings and their computation, such as elliptic curves, their group law, pairings and pairing-friendly elliptic curves. To ease the exposition, the material is not presented in the most general setting, but rather focused on special cases that are most common and representative of the general ideas. For more details, we point the reader to [60, 13, 25, 26, 32].

9.1.1 Elliptic curves

Let $p > 3$ be a prime and $\mathbb{F}_q$ be a finite field of characteristic p. Throughout the whole chapter, we consider E to be an elliptic curve defined over $\mathbb{F}_q$, given by a short Weierstrass equation $E : y^2 = x^3+ax+b$, where $a, b \in \mathbb{F}_q$ and $4a^3+27b^2 \neq 0$. The point at infinity on E is denoted by O. The curve E can be viewed as the set of affine solutions (x, y) of the curve equation with coordinates in an algebraic closure of $\mathbb{F}_q$ together with the special point O. The set of $\mathbb{F}_q$-rational points on E is given by $E(\mathbb{F}_q) = \{(x, y) \in \mathbb{F}_q \times \mathbb{F}_q \mid y^2 = x^3 + ax + b\} \cup \{O\}$, and the number of such points is $n = \#E(\mathbb{F}_q) = q + 1 - t$, where t is bounded by $|t| \leq 2\sqrt{q}$.

The group law

There exists an abelian group law $+$ on E, which is given as follows. The point O is the neutral element, i.e., $P_1+O = P_1$ for any point $P_1 \in E$. If $P_1 \neq O$, write $P_1 = (x_1, y_1)$. Then, the inverse element of P_1 is the point $-P_1 = (x_1, -y_1)$, i.e., $(x_1, y_1) + (x_1, -y_1) = O$. Given another point $P_2 = (x_2, y_2) \in E \setminus \{O, -P_1\}$, define

$$\lambda = \begin{cases} (y_2 - y_1)/(x_2 - x_1) & \text{if } P_1 \neq P_2, \\ (3x_1^2 + a)/(2y_1) & \text{if } P_1 = P_2. \end{cases} \tag{9.1}$$

Then $P_3 = (x_3, y_3) = P_1+P_2$ is given by $x_3 = \lambda^2-x_1-x_2$ and $y_3 = \lambda(x_1-x_3)-y_1$.

Computing the sum of two affine $\mathbb{F}_q$-rational points requires a division in the field $\mathbb{F}_q$. This operation is usually significantly more costly than all the other operations, such as addition, subtraction and multiplication. Therefore, elliptic curve group operations are often computed using a projective coordinate system, in which an additional third coordinate keeps track of all denominators such that field inversions are avoided altogether until the very end of the computation. This comes at the cost of additional field multiplications and other lower-cost operations. There are various forms of projective coordinate systems suitable for different scenarios.

Embedding degree and torsion points

Let r be a prime with $r \mid n$, and 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 the multiplicative group $\mathbb{F}_{q^k}^*$ contains as a subgroup the group μ_r of r-th roots of unity. The embedding degree k 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 $[m] : E \to E$ be the multiplication-by-m map, which maps a point $P \in E$ to the sum of m copies of P. 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 ($\ell > 0$). From now on it is assumed that $k > 1$, in which case $E[r] \subseteq E(\mathbb{F}_{q^k})$, i.e., all r-torsion points are defined over $\mathbb{F}_{q^k}$.

9.1.2 Pairings

In order to describe the pairing functions relevant to cryptography, we need to introduce divisors on E. A divisor D on E is a formal sum of points with integer coefficients, i.e., $D = \sum_{P \in E} n_P(P)$ with only finitely many of the n_P different from zero; this finite set of non-zero n_P-values is the support of the divisor D. A divisor is called a principal divisor, if it is the divisor of a function f on E. In that case the coefficient n_P reflects the multiplicity of P as a zero (if $n_P > 0$) or pole ($n_P < 0$) of f. Two divisors are equivalent if they differ by a principal divisor. For $P_1 \in E$, let D_{P_1} be a divisor that is equivalent to $(P_1) - (\mathcal{O})$. Let f_{r,P_1} be a function on E with divisor $r(P_1) - r(\mathcal{O})$. Evaluating a function f at a divisor $\sum n_P(P)$ means to evaluate $\prod f(P)^{n_P}$.

The Weil pairing

The first applications of pairings in cryptography used the Weil pairing

$$e_r = E[r] \times E[r] \to \mu_r \subseteq \mathbb{F}_{q^k}^*,\ (P, Q) \mapsto f_{r,P}(D_Q)/f_{r,Q}(D_P).$$

For the computation of $f_{r,Q}(D_P)$, we can take $D_Q = (Q) - (\mathcal{O})$ and need to choose a suitable point R such that $D_P = (P + R) - (R)$ has support disjoint from $\{\mathcal{O}, Q\}$ (and similarly for $f_{r,P}(D_Q)$).

The Tate pairing

Most pairings that are suitable for use in practical cryptographic applications are derived from the reduced Tate pairing

$$t_r = E(\mathbb{F}_q)[r] \times E(\mathbb{F}_{q^k})[r] \to \mu_r \subseteq \mathbb{F}_{q^k}^*,\ (P, Q) \mapsto f_{r,P}(D_Q)^{(q^k-1)/r}.$$

In the case $k > 1$, to which we restrict in this chapter, one can omit the auxiliary point in the divisor D_Q and compute $f_{r,P}(Q)^{(q^k-1)/r}$ instead (see [8, 9, Thm. 1]).

Miller's algorithm

To evaluate the functions occurring in the Weil and the Tate pairing, one follows an iterative approach based on Miller's formulas [50]. For an integer $m \in \mathbb{Z}$ and a point $P \in E(\mathbb{F}_{q^k})[r]$ define the function $f_{m,P}$ to be a function with divisor $m(P) - ([m]P) - (m-1)(\mathcal{O})$. Note that this notation coincides with the one used above for $m = r$ because then $[r]P = \mathcal{O}$. The value of the function $f_{r,P}$ at the point Q can be computed in a square-and-multiply-like fashion via function values $f_{m,P}(Q)$ for $m < r$ with the help of certain line functions. Let $P_1, P_2, Q \in E$ such that $P_1 \neq \mathcal{O}$, $P_2 \notin \{\mathcal{O}, -P_1\}$, and $E \neq \mathcal{O}$. Then the line through $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$, evaluated at $Q = (x_Q, y_Q)$ is given by

$$l_{P_1,P_2}(Q) = (y_Q - y_1) - \lambda(x_Q - x_1),$$

where λ is the value of the slope computed during the computation of $P_1 + P_2$ as in Equation (9.1) on page 3. In the case that $P_2 = -P_1$, we have the vertical line function value $v_{P_1}(Q) = l_{P_1,-P_1}(Q) = x_Q - x_1$. Define the quotient $g_{P_1,P_2}(Q) = l_{P_1,P_2}(Q)/v_{P_1+P_2}(Q)$. Then, for $m_1, m_2 \in \mathbb{Z}$, Miller's formulas hold as follows:

$$\begin{aligned} f_{m_1+m_2,P}(Q) &= f_{m_1}(Q) f_{m_2}(Q) g_{[m_1]P,[m_2]P}(Q), \\ f_{m_1 m_2,P}(Q) &= f_{m_1}^{m_2}(Q) f_{m_2,[m_1]P}(Q) = f_{m_2}^{m_1}(Q) f_{m_1,[m_2]P}(Q). \end{aligned}$$

Specializations of these formulas yield $f_{m+1,P}(Q) = f_{m,P}(Q) g_{[m]P,P}(Q)$, and $f_{2m,P}(Q) = f_{m,P}^2(Q) g_{[m]P,[m]P}(Q)$, which lead to Miller's algorithm to compute $f_{r,P}(Q)$ as shown in Algorithm 1 on page 5. Note that as written here, in the last iteration of the algorithm, since $r_0 = 1$, the point addition in line 5 computes the point at infinity $\mathcal{O}$ in an exceptional case. The value of the function $g_{R,P}$ in this step is defined as $g_{R,P}(Q) = v_P(Q)$.

Algorithm 1 Miller's algorithm

Input: $P, Q \in E[r], r = (r_{\ell-1}, r_{\ell-2}, \ldots, r_0)_2, r_{\ell-1} = 1$
Output: $f_{r,P}(Q)$ representing a class in $\mathbb{F}_{q^k}^*/(\mathbb{F}_{q^k}^*)^r$

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

Unlike scalar multiplication algorithms for exponentiation in elliptic curve

groups, which multiply curve points by varying scalars that are often secret values in the respective cryptographic protocols, Miller's algorithm works alongside a scalar multiplication with a fixed scalar, namely r. In other words, the exponentiation carried out in Miller's algorithm is always to the same public number. Significant savings can be obtained by choosing this system parameter with a low Hamming weight (or low weight in a signed binary representation) in order to have as few as possible of the addition steps in line 5 of Algorithm 1.

The final exponentiation to the power $(q^k - 1)/r$ in the Tate pairing after the Miller loop represents a large part of the computation. It maps classes in $\mathbb{F}_{q^k}^*/(\mathbb{F}_{q^k}^*)^r$ to unique representatives in μ_r and is an exponentiation with a known fixed, special exponent. Parts of it are carried out in special subgroups of the multiplicative group $\mathbb{F}_{q^k}^*$. This leads to various improvements over a general exponentiation with an exponent of that size, leading to a significant speedup (see, for example [59, 35, 39]).

The ate pairing

Often the most efficient choices for implementation are variants of the ate pairing [41], which is a certain power of the reduced Tate pairing and is defined on special subgroups of $E[r]$.

Let ϕ_q be the q-power Frobenius endomorphism on E, i.e., $\phi_q(x, y) = (x^q, y^q)$. 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 group G_1 contains only points defined over the base field $\mathbb{F}_q$, while the points in G_2 are minimally defined over $\mathbb{F}_{q^k}$. The *ate pairing* is the map

$$a_T : G_2 \times G_1 \to \mu_r, \quad (Q, P) \mapsto f_{T,Q}(P)^{(q^k-1)/r}, \tag{9.2}$$

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 [41]). 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 $G_2' \times G_1 \to \mu_r,\ (Q', P) \mapsto f_{T,\psi(Q')}(P)^{(q^k-1)/r}$.

Algorithm 2 shows Miller's algorithm in the just described setting for an ate-like pairing to compute $f_{m,\psi(Q')}(P)$ for some integer $m > 0$. In this algorithm and throughout the rest of this chapter, we assume that k is even and therefore the denominator elimination technique applies (see [8, 9]), which means that the denominators in the above defined fraction $g_{Q_1',Q_2'}(P)$ of line functions can be omitted without changing the pairing value. This is why Algorithm 2 only uses the line functions $l_{Q_1',Q_2'}(P)$.

Algorithm 2 Miller's algorithm for even k and ate-like pairings

Input: $Q' \in G_2', P \in G_1, m = (m_{l-1}, m_{l-2}, \ldots, m_0)_2, m_{l-1} = 1$
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 $f_{m,\psi(Q')}(P)$ 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 at $R = \psi(R')$, and doubling the point R'. Similarly, an *addition step* is computed in step 5 of Algorithm 2.

The most efficient variants of the ate pairing are so-called optimal ate pairings [61]. 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 and log is the logarithm to base 2. For such 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 (see [61] for details).

9.1.3 Pairing-friendly elliptic curves

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 [29]. For security reasons, the parameters need to have certain minimal sizes which imply suitable values for the embedding degree k for specific security levels.

The requirement for an elliptic curve to have small embedding degree is a strong condition that restricts the set of possible curves to a small subset of elliptic curves. Because of the possibility of a transfer attack on the discrete logarithm problem like the Menezes-Okamoto-Vanstone [49] or the Frey-Rück [31, 30] attacks, such curves are excluded from being used in elliptic curve-based protocols that do not require a pairing function. In general, an arbitrary elliptic curve over a finite field with a large prime divisor of the group order will have a large embedding degree. This means that pairing-friendly curves are special and rare and finding a curve that satisfies the required properties can be a challenge.

The most popular choices for pairing-friendly curves come from polynomial families of curves. In these families, the base field prime and the prime group order are parameterized by rational polynomials that ensure all conditions are satisfied. Concrete parameters are obtained by searching for an integer parameter such that the above polynomials evaluate to prime integers at the parameter. A corresponding elliptic curve can then be constructed using the complex multiplication method, or a simpler algorithm that tests for the correct group order in certain special cases.

As explained in subsection 9.1.2 on page 6, it is often 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 advantageous. 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 $\mathbb{F}_q$.

9.2 Finite field arithmetic for pairings

This section takes a closer look at the field arithmetic needed for pairing algorithms. As above, let $\mathbb{F}_q$ be a finite field of characteristic $p > 3$, and let $\mathbb{F}_{q^m}$ be an extension of degree m of $\mathbb{F}_q$ for $m \mid k$. Note that mostly, pairing-friendly elliptic curves are defined over a prime field, i.e., $q = p$. Algorithm 2 on page 7 shows that a pairing algorithm requires arithmetic in the base field and also in

some of the field extensions $\mathbb{F}_{q^m}$. A crucial building block with major influence on the overall efficiency of pairing-based protocols is multiplication modulo the prime p. An obvious influence of Peter's work is the use of Montgomery arithmetic for modular operations.

Another aspect that has been inspired by Peter's way of thinking is the treatment of inversions. As indicated above, usually the cost for a field inversion is larger than that for a field multiplication, which in turn is larger than that for an addition and a subtraction. What the exact ratios between these are, depends on the setting, i.e., the shape of the base field prime, the extension degree, the algorithms used, and the computer architecture they are implemented for. When writing code that has to have a constant execution time, independent of secret input data, inversions are often even computed using a field exponentiation based on Fermat's little theorem. Thus, in implementations of prime fields, inversions are usually very expensive, in which case it does not make sense to use affine coordinates. Pairing implementations using some form of projective coordinates can get away with a single inversion in a pairing evaluation. But still, there are certain scenarios discussed later, in which it is more efficient to work in affine coordinates when doing the elliptic curve operations during the pairing algorithm. Furthermore, divisions are still needed, for example, to scale curve points to a unique affine representation in the non-pairing operations. After a brief discussion on multiplication, this section describes in some detail the circumstances that allow to achieve relatively low inversion costs.

9.2.1 Montgomery multiplication

The base field primes for pairing-friendly elliptic curves can generally not be chosen to have a specific implementation-friendly form as is done for plain elliptic curve cryptography to obtain faster modular multiplications. For example, elliptic curve implementations for key exchange or digital signatures often use curves defined over fields with special prime shapes such as Solinas primes, Montgomery-friendly primes or pseudo-Mersenne primes. Such choices make modular reduction very efficient. Since the parameters of pairing-friendly curves have to satisfy the additional embedding degree condition, adding a condition on the prime shape would be too restrictive. Therefore, base field primes for those curves do not have a special structure that can be exploited to speed up modular reduction.

As a consequence, Montgomery multiplication and Montgomery reduction (see Chapter 2) are a good choice for efficient implementation of modular arithmetic, and are the method used in speed-record pairing implementations [12, 3, 1].

9.2.2 Multiplication in extension fields

The field extension $\mathbb{F}_{q^k}$ required in the pairing algorithm is usually constructed as a tower of small-degree field extensions (preferably extensions of degree 2 and 3, see the notion of pairing-friendly fields in [47]), depending on the factorization of the embedding degree k. Benger and Scott [10] discuss how to best choose such towers in the pairing setting. Multiplication in each intermediate extension is implemented using the Karatsuba or the Toom-Cook method [47].

The line function values that occur in Miller's algorithm when using the group G_2' on a high-degree twist are sparse elements of $\mathbb{F}_{q^k}$, i.e., some of their coefficients when written as a polynomial over $\mathbb{F}_q$ are always zero. Therefore, multiplications with these values can exploit special, more efficient multiplication routines.

9.2.3 Finite field inversions

Itoh and Tsujii [43] describe a method for computing the inverse of an element in a binary field using normal bases and Kobayashi et al. [46] 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 [6, 38] and [24, Sections 11.3.4 and 11.3.6]).

It can be applied in the following setting. Let $\mathbb{F}_{q^\ell} = \mathbb{F}_q(\alpha)$ 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. Let $\mathbf{M}_{q^m}$, $\mathbf{S}_{q^m}$, $\mathbf{I}_{q^m}$, $\mathbf{add}_{q^m}$, $\mathbf{sub}_{q^m}$, and $\mathbf{neg}_{q^m}$ denote the costs for multiplication, squaring, inversion, addition, subtraction, and negation in the field $\mathbb{F}_{q^m}$. 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. Denote the inversion to multiplication cost ratio by $\mathbf{R}_{q^m} = \mathbf{I}_{q^m}/\mathbf{M}_{q^m}$.

A Frobenius computation using a look-up table of $\ell - 1$ precomputed values in $\mathbb{F}_q$ consisting of powers of ω costs at most $\ell - 1$ multiplications in $\mathbb{F}_q$ (see

[46, Section 2.3], note $\gcd(\ell, q) = 1$). According to [40, Section 2.4.3] the computation of β^{v-1} via an addition chain approach, using a look-up table for each required 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 the following upper bound for the cost of an inversion in $\mathbb{F}_{q^\ell}$:

$$\begin{aligned}\mathbf{I}_{q^\ell} \leq \mathbf{I}_q + (\lfloor \log(\ell - 1) \rfloor + h(\ell - 1))(\mathbf{M}_{q^\ell} + (\ell - 1)\mathbf{M}_q) \\ +2\ell\mathbf{M}_q + \mathbf{M}_{(\omega)} + (\ell - 1)\mathbf{add}_q. \qquad (9.3)\end{aligned}$$

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

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 (9.3), 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 9.1 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 [53].

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

Table 9.1 *Constants that determine the improvement of* $\mathbf{R}_{q^\ell}$ *over* $\mathbf{R}_q$

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

Example 9.2 (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 one inversion, two multiplications, two squarings, one multiplication by ω, one subtraction and one negation, all in $\mathbb{F}_q$, i.e., $\mathbf{I}_{q^2} = \mathbf{I}_q + 2\mathbf{M}_q + 2\mathbf{S}_q + \mathbf{M}_{(\omega)} + \mathbf{sub}_q + \mathbf{neg}_q$.

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 we can give an upper bound on the $\mathbf{I}/\mathbf{M}$-ratio in $\mathbb{F}_{q^2}$ depending on the ratio in $\mathbb{F}_q$ as

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

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 $\mathbf{R}_q/3$.

Example 9.3 (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 two-way 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 three more multiplications and two 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

$$\mathbf{R}_{q^3=}\mathbf{I}_{q^3}/\mathbf{M}_{q^3} \leq (\mathbf{I}_q/\mathbf{M}_q)/6 + 3 = \mathbf{R}_q/6 + 3.$$

Inversions in towers of field extensions

Baktir and Sunar [7] 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. For towers of field extensions of degree two and three as those occurring in pairing-friendly fields, the inversion can be done using the two examples given above.

9.2.4 Simultaneous inversions

The inverses of s field elements $a_1, \ldots, a_s$ can be computed simultaneously with Montgomery's well-known inversion-sharing trick [52, Section 10.3.1.] at the cost of one 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 $b^{-1} = a\cdot(ab)^{-1}$, with an overall cost of just one inversion and three multiplications.

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 [24, Algorithm 11.15]), where we need $2(s-1)$ more multiplications to get the inverses of all elements.

Since this method works for general field elements, i.e., it is not restricted to a specific extension degree, we leave out the indices in the notation of the inversion and multiplication costs and their ratio. The cost for s inversions is replaced by $\mathbf{I} + 3(s-1)\mathbf{M}$. Let $\mathbf{R}_{\mathrm{avg},s}$ denote the ratio of the cost of s inversions to the cost of s multiplications. It is bounded above by

$$\mathbf{R}_{\mathrm{avg},s} = \mathbf{I}/(s\mathbf{M}) + 3(s-1)/s \leq \mathbf{R}/s + 3,$$

i.e., when the number s of elements to be inverted grows, the ratio $\mathbf{R}_{\mathrm{avg},s}$ gets closer to 3. Note that most of the time, this method improves the efficiency of an implementation whenever applicable. However, in large field extensions,

the inversion method described above in Section 9.2.3 on page 10 might lead to an **I**/**M**-ratio that already is less than 3. In this case using the sharing trick at the highest level of the field extension would make the average ratio worse. But a combination of both methods, such as reducing the inversions down to the ground field and sharing them there, can further increase efficiency.

9.3 Affine coordinates for pairing computation

The previous section elaborated on two techniques that achieve a low inversion to multiplication cost ratio, namely working in extension fields and doing several inversions at once if possible. There exist scenarios in which one of the two or both techniques can be applied and lead to affine coordinates being the most efficient choice. Affine coordinates are useful for curve operations in the group G_2' including those needed in the Miller loop, whenever a pairing-friendly curve is chosen such that G_2' is defined over an extension field of larger degree. This might occur for high security pairings or whenever high-degree twists cannot be used. The second technique of simultaneous inversion becomes possible whenever several pairings or a product of several pairings is computed such that inversions can be synchronized and be done simultaneously. This section states the costs for the doubling and addition steps in Miller's algorithm for affine coordinates and explains use cases of the above two techniques for achieving low inversion cost.

9.3.1 Costs for doubling and addition steps

We begin by describing the evaluation of line functions in affine coordinates for ate-like pairings as in Section 9.1.2 on page 6, i.e., a point P on E, $P \neq O$, 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 $R_1, R_2, S \neq O$. 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 $l_{R_1,R_2}(S) = x_S - x_{R_1}$.

Let the notation be as described in Section 9.1.2, in particular we use a twist E' of degree d to represent the group G_2 by the group G_2'. 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 the field extension $\mathbb{F}_{q^k}$ is given by $\mathbb{F}_{q^k} = \mathbb{F}_{q^e}(\alpha)$ 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 $[2]R' \neq O$ 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 9.2 on page 16. 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 formulas using projective coordinates taken from [42] and [23]. For an overview of the most efficient explicit formulas known for elliptic-curve operations see the Explicit-Formulas Database [11]. We transfer the formulas in [42] to the ate pairing using the trick in [23] 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 chapter. Still, in the case $d = 2$, evaluation of the line function cannot be done in $k\mathbf{M}_q$; instead two multiplications in $\mathbb{F}_{q^{k/2}}$ need to be done (see also the discussion in the respective sections of [23]). Furthermore, we assume that all precomputations are done as described in the above papers and small multiples are computed by additions.

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

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

Table 9.2 *Operation counts for the doubling step in the ate-like Miller loop omitting* $1\mathbf{S}_{q^k} + 1\mathbf{M}_{q^k}$. *The elements* $a, b \in \mathbb{F}_q$ *are the curve coefficients,* k *is the embedding degree with respect to the prime subgroup order* r, *and* d *is the degree of the twist such that the group* G_2' *is defined over* $\mathbb{F}_{q^e}$ *where* $e = k/d$.

DBL	d	coord.	$\mathbf{M}_q$	$\mathbf{I}_{q^e}$	$\mathbf{M}_{q^e}$	$\mathbf{S}_{q^e}$	$\mathbf{M}_{(\cdot)}$	$\mathbf{add}_{q^e}$	$\mathbf{sub}_{q^e}$	$\mathbf{neg}_{q^e}$
$ab \neq 0$	2	affine	$k/2$	1	3	2	$1\mathbf{M}_{(\omega)}$	4	6	–
$2 \mid k$		Jac. [42]	–	–	3	11	$1\mathbf{M}_{(a/\omega^2)}$	6	17	–
$b = 0$	4	affine	$k/4$	1	3	2	–	4	5	1
$4 \mid k$		$W_{(1,2)}$ [23]	$k/2$	–	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. [23]	$k/3$	–	2	7	$1\mathbf{M}_{(b/\omega)}$	11	10	1

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 9.3 compares the costs for affine addition steps to those in projective coordinates. Again, we take these operation counts from the literature (see [4, 23, 22] 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 9.3.

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 9.3 shows that the cost traded for the inversion in the projective addition formulas is equivalent to at least several $\mathbf{M}_{q^e}$. For doubling steps, it is smaller, but still equivalent to a few $\mathbf{M}_{q^e}$ 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

Table 9.3 *Operation counts for the addition step in the ate-like Miller loop omitting* $1\mathbf{M}_{q^k}$. *The elements* $a, b \in \mathbb{F}_q$ *are the curve coefficients,* k *is the embedding degree with respect to the prime subgroup order* r*, and* d *is the degree of the twist such that the group* G_2' *is defined over* $\mathbb{F}_{q^e}$ *where* $e = k/d$.

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 [4]	–	–	8	6	6	17	–
$b = 0$	4	affine	$k/4$	1	3	1	–	7	1
$4 \mid k$		$W_{(1,2)}$ [23]	$k/2$	–	9	5	7	8	1
$a = 0$	6	affine	$k/6$	1	3	1	–	7	1
$6 \mid k$		proj. [22, 23]	$k/3$	–	11	2	1	7	–

function is chosen), the traded cost in the doubling case is the most relevant to consider.

The subsection concludes with two examples that give specific upper bounds on $\mathbf{I}_{q^e}$ such that affine coordinates are more efficient than projective ones.

Example 9.4 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 $\mathbf{add}_{q^{k/2}} \approx \mathbf{sub}_{q^{k/2}}$. We see that if an inversion costs less than $8\mathbf{S}_{q^{k/2}} + 13\mathbf{add}_{q^{k/2}}$, then affine coordinates are better than Jacobian.

Example 9.5 In the case $a = 0$, $d = 6$, similar to the previous example, we deduce that if an inversion in $\mathbb{F}_{q^{k/6}}$ is less than $5\mathbf{S}_{q^{k/6}} - \mathbf{M}_{q^{k/6}} + (k/6)\mathbf{M}_q + \mathbf{M}_{(b/\omega)} + 12\mathbf{add}_{q^{k/6}}$, then affine coordinates beat the projective ones.

In order to fully assess the effect of using affine instead of projective coordinates, one needs to know the exact cost that is traded for the inversion. This strongly depends on the specific algorithms chosen to implement the operations in the extension fields. For example, the relative costs between multiplications and squarings can differ significantly. Commonly used values in the literature are $\mathbf{S}_q = \mathbf{M}_q$ or $\mathbf{S}_q = 0.8\mathbf{M}_q$ when q is a prime, see [11]. Using Karatsuba multiplication and squaring in a quadratic extension leads to a ratio of $\mathbf{S}_{q^2} \approx (2/3)\mathbf{M}_{q^2}$. Therefore, we do not specify these costs any further and leave the evaluation for a point when a specific scenario and an implementation is chosen.

9.3.2 Working over extension fields

This subsection illustrates the usefulness of affine coordinates when the extension degree e is relatively large.

To implement pairings at a given security level, on the one hand one needs to find a pairing-friendly elliptic curve such that r and q^k have a certain minimal size (e.g. given by current estimates for the runtimes of algorithms to solve the ECDLP). On the other hand, for efficiency it is desirable that they are not much larger than necessary. 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. The value ρk is the ratio of the sizes of q^k and r. For a given curve the value ρk describes how well security is balanced between the curve groups and the finite field group.

An overview of construction methods for pairing-friendly elliptic curves is given in [29]. In Table 9.4, we list suggestions for curve families by their construction in [29] for high-security levels of 128, 192, and 256 bits. The last column in Table 9.4 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 9.4.

Remark 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 9.4). 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 9.1 on page 12). The costs for doubling and addition steps given in the second lines of Tables 9.2 on page 16 and 9.3 on the previous page for degree-4 twists show that the cost of the inversion avoided in a projective implementation should be compared with roughly $6\mathbf{S}_{q^7} + 5\mathbf{add}_{q^7} +$

Table 9.4 *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.*

security	construction in [29]	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	$\mathbb{F}_{p^{13}}$

$5\mathbf{sub}_{q^7}$ 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.

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 9.2 and 9.3 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 9.4, so we have extension degree 6 for 128-bit security up to extension degree 13 for 256-bit security.

9.3.3 Simultaneous inversions in pairing computation

This subsection discusses different scenarios, in which the simultaneous inversion technique from Section 9.2.4 on page 13 can be applied and might lead to a low-enough $\mathbf{I}/\mathbf{M}$ ratio to favor affine coordinates.

Sharing inversions in a single pairing computation

Schroeppel and Beaver [56] demonstrate the use of the inversion-sharing trick to speed up a single scalar multiplication on an elliptic curve in affine coordi-

nates. 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 3. Unfortunately, addition steps cost much more than in the conventional left-to-right algorithm as it is given in Algorithm 2 on page 7. 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 cannot be compensated for by using affine coordinates with the inversion-sharing trick.

Algorithm 3 Right-to-left version of Miller's algorithm with postponed addition steps for even k and ate-like pairings

Input: $Q' \in G_2', P \in G_1, m = (m_{l-1}, m_{l-2}, \ldots, m_0)_2, m_{l-1} = 1$
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

Parallelizing a single pairing

However, the right-to-left algorithm can be parallelized, and this could lead to more efficient implementations by taking advantage of many-core machines. Grabher, Großschädl, and Page [34, Algorithm 2] use a version of Algorithm 3

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

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. The communication overhead in order to gather data from intermediate computations on different cores to compute the inverse at one core imposes a non-trivial performance penalty and it remains to be seen whether this approach is worthwhile. Grabher et al. [34] note that when multiple pairings are computed, it is better to parallelize by performing one pairing on each core.

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. [15]; and the non-interactive proof systems proposed by Groth and Sahai [37] 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 [57, Section 4.3] and Granger and Smart [36]. See also the more recent work [58] by Scott.

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 sufficiently many pairings so 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 9.3.1 on page 14.

Products of pairings

For computing a product of pairings, more optimizations can be applied, including the above inversion-sharing. Scott [57, 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 [36] 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.

9.4 The double-add operation and parabolas

This section consists largely of lightly edited material from Sections 3, 5, and 6 of [27]. In [27], an improvement to the double-and-add operation on an elliptic curve was introduced, and a related idea was applied to improve the analogous operation in pairing computations on elliptic curves. In affine coordinates, the technique allows one to perform a doubling and an addition, $2P + Q$, on an elliptic curve E using only $1\mathbf{M} + 2\mathbf{S} + 2\mathbf{I}$ (plus an extra squaring when $P = Q$). This is achieved as follows: to compute $2P + Q$, where $P = (x_1, y_1)$ and $Q = (x_2, y_2)$, first find $(P + Q)$, but omit its y-coordinate since it is not needed for the next stage. This saves one field multiplication. Next compute $(P+Q)+P$. This way, two point additions are performed while saving one multiplication. This trick also works when $P = Q$, i.e., when tripling a point.

9.4.1 Description of the algorithm

Suppose $P = (x_1, y_1)$ and $Q = (x_2, y_2)$ are distinct points on E different from O, and $x_1 \neq x_2$. The details for other cases are given in Figure 1 of [27]. That figure also covers special cases, where an input or an intermediate result is the point O. Recall from Section 9.1.1 and Equation (9.1) on page 3

that the point $P + Q$ is then given by $(x_3,\ y_3)$, where $x_3 = \lambda_1^2 - x_1 - x_2$ and $y_3 = (x_1 - x_3)\lambda_1 - y_1$ with $\lambda_1 = (y_2 - y_1)/(x_2 - x_1)$.

To add $(P + Q)$ to P, add $(x_1,\ y_1)$ to $(x_3,\ y_3)$ using the above rule (assuming $x_3 \neq x_1$). The result has coordinates $(x_4,\ y_4)$, where $x_4 = \lambda_2^2 - x_1 - x_3$ and $y_4 = (x_1 - x_4)\lambda_2 - y_1$ with $\lambda_2 = (y_3 - y_1)/(x_3 - x_1)$.

The computation of y_3 can be omitted because it is used only in the computation of λ_2, which can be computed without knowing y_3 as

$$\lambda_2 = -\lambda_1 - 2y_1/(x_3 - x_1).$$

Omitting the y_3 computation saves one field multiplication. Each λ_2 formula requires a field division, so the overall saving is this one field multiplication.

This trick can also be applied to save one multiplication when computing $3P$, the triple of a point $P \neq O$, where the λ_2 computation will need the slope of a line through two distinct points $2P$ and P. It can be used twice to save two multiplications when computing $3P+Q = ((P+Q)+P)+P$. Thus $3P+Q$ can be computed using one multiplication, three squarings, and three divisions. Such a sequence of operations would be performed repeatedly if an exponent were written in ternary form and left-to-right exponentiation were used. Ternary representation performs worse than binary representation for large random exponents k, but the operation of triple and add was explored further in the context of double-base chains in [21, Section 5].

9.4.2 Application to scalar multiplication

The above double-and-add operation can be applied to scalar multiplications on an elliptic curve in affine coordinates. In the naive left-to-right method of binary exponentiation, the double-and-add trick can be applied repeatedly at each stage of the computation. More efficient variants could use a combination of the left-to-right or right-to-left m-ary exponentiation with sliding window methods, addition-subtraction chains, signed representations, etc. When using affine coordinates, the double-and-add trick can be used on top of these methods for $m = 2$ to obtain savings, depending upon the window size that is used.

Note that we discuss the double-and-add technique in a historic manner and neglect the protection of scalar multiplication against side channels such as making sure that the algorithm runs in constant time.

Another application, in which the double-and-add operation occurs more frequently than in single scalar multiplication is multi-exponentiation, for example a simultaneous three-fold scalar multiplication $k_1P_1+k_2P_2+k_3P_3$, where the three exponents k_1, k_2, and k_3 have approximately the same length. One algorithm creates an 8-entry table with the precomputed points $O, P_1, P_2, P_2 +$

$P_1, P_3, P_3 + P_1, P_3 + P_2, P_3 + P_2 + P_1$. Subsequently it uses one elliptic curve doubling followed by the addition of a table entry, for each bit in the exponents [51]. For random scalars k_i and a binary scalar decomposition, about $7/8$ of the doublings will be followed by an addition other than O.

9.4.3 Application to pairings

The double-and-add technique can be extended to the function evaluation in Miller's algorithm and thus can be applied to pairing computation as well. The key observation is that one can replace consecutive evaluations of line functions by the evaluation of a parabola, and thus obtain an analogue to the double-and-add operation in the form of a Miller formula.

Using the double-and-add trick with parabolas

We use notation as in the paragraph on Miller's algorithm in Section 9.1.2 on page 4. For two integers b, c, instead of computing $f_{2b+c,P}(Q)$ and $[2b + c]P$ from $(f_{b,P}(Q), [b]P)$ and $(f_{c,P}(Q), [c]P)$ via $f_{2b,P}(Q) = f_{b,P}^2(Q)g_{[b]P,[b]P}$ and $[2]([b]P)$, and then $f_{2b+c,P}(Q) = f_{2b,P}(Q)f_{c,P}(Q)g_{[2b]P,[c]P}$ and $[2b]P + [c]P$, we now describe an alternative approach that computes the result directly, producing only the x-coordinate of the intermediate point $[b]P + [c]P$. To combine the two steps, one constructs a parabola through the points $[b]P$, $[b]P$, $[c]P$, $-[2b + c]P$.

To form $f_{2b+c,P}(Q)$, combine forming $f_{b+c,P}(Q)$ with $f_{b+c+b,P}(Q)$ as follows. We omit the notation for evaluation at Q and simply write f_{2b+c} etc. and use the notation l_P for the vertical line function $l_{P,-P}$ through P and $-P$. We have

$$\begin{aligned} f_{2b+c,P} &= f_{b+c,P} \cdot f_{b,P} \cdot g_{[b+c]P,[b]P} \\ &= (f_{b,P} \cdot f_{c,P} \cdot g_{[b]P,[c]P}) \cdot (f_{b,P} \cdot g_{[b+c]P,[b]P}) \\ &= \frac{f_{b,P}^2 \cdot f_{c,P}}{l_{[2b+c]P}} \cdot \frac{l_{[b]P,[c]P} \cdot l_{[b+c]P,[b]P}}{l_{[b+c]P}}. \end{aligned}$$

Now, one can replace the second fraction $l_{[b]P,[c]P} \cdot l_{[b+c]P,[b]P}/l_{[b+c]P}$ by the parabola whose formula is given in the next paragraph.

Equation for the parabola

If R and S are points on E, then there is a (possibly degenerate) parabolic equation passing through R twice (i.e., tangent at R) and also passing through S and $-[2]R - S$. Using the notation $R = (x_1, y_1)$ and $S = (x_2, y_2)$ with $R + S = (x_3, y_3)$ and $2R + S = (x_4, y_4)$, a formula for this parabola is

$$\frac{(y + y_3 - \lambda_1(x - x_3))(y - y_3 - \lambda_2(x - x_3))}{x - x_3}. \tag{9.4}$$

The left half of the numerator of (9.4) is a line passing through R, S, and $-R-S$ whose slope is λ_1. The second half of the numerator is a line passing through $R+S$, R, and $-[2]R-S$, whose slope is λ_2. The denominator is a (vertical) line through $R+S$ and $-R-S$. The quotient has zeros at R, R, S, $-[2]R-S$ and a pole of order four at O.

One can simplify Equation (9.4) by expanding it in powers of $x-x_3$, using the Weierstrass equation for E to eliminate references to y^2 and y_3^2.

$$\begin{aligned}
&\frac{y^2-y_3^2}{x-x_3} - \lambda_1(y-y_3) - \lambda_2(y+y_3) + \lambda_1\lambda_2(x-x_3) \\
&= x^2 + xx_3 + x_3^2 + a + \lambda_1\lambda_2(x-x_3) - \lambda_1(y-y_3) - \lambda_2(y+y_3) \qquad (9.5)\\
&= x^2 + (x_3+\lambda_1\lambda_2)x - (\lambda_1+\lambda_2)y + (x_3^2 + a - \lambda_1\lambda_2 x_3 + (\lambda_1-\lambda_2)y_3).
\end{aligned}$$

Knowing that (9.5) passes through $R=(x_1,y_1)$, one succinct formula for the parabola is

$$(x-x_1)(x+x_1+x_3+\lambda_1\lambda_2) - (\lambda_1+\lambda_2)(y-y_1). \qquad (9.6)$$

In the previous section we can now replace $l_{[b]P,[c]P} \cdot l_{[b+c]P,[b]P}/l_{[b+c]P}$ by the parabola (9.6) with $R=[b]P$ and $S=[c]P$. Formula (9.6) for the parabola does not reference y_3 and is never identically zero since its x^2 coefficient is 1.

The following example shows how in the case of affine coordinates, the double-and-add operation can help to make the pairing algorithm more efficient.

Example 9.6 Assume that we are computing an ate-like pairing in the same setting as in Section 9.3.1 on page 14. Let $ab \neq 0$, i.e., $d=2$, let k be even such that we can use denominator elimination, and note that $\omega = \alpha^2$. Now, assume that we are at a position in the Miller loop, where we are computing a doubling step followed by an addition step. This means that the two steps compute $f_{[2]R'+Q',\psi(Q')}(P)$ from $f_{R',\psi(Q')}(P)$ and R'. Let us first analyse the cost when doing this the usual way. We simply add the costs for doubling and addition from Tables 9.2 on page 16 and 9.3 on page 17, respectively. The resulting cost is $k\mathbf{M}_q + 2\mathbf{I}_{q^{k/2}} + 6\mathbf{M}_{q^{k/w}} + 3\mathbf{S}_{q^{k/2}} + 1\mathbf{M}_{(\omega)} + 4\mathbf{add}_{q^{k/2}} + 14\mathbf{sub}_{q^{k/2}}$ plus the squaring cost $\mathbf{S}_{q^k}$ for the standard Miller loop squaring and two multiplications $2\mathbf{M}_{q^k}$ with line functions, which are regular field multiplications in $\mathbb{F}_{q^k}$ because line functions are not sparse in this case.

Next assume that we are using the parabola technique instead of the usual two steps. The squaring cost $\mathbf{S}_{q^k}$ in the large field remains, but we only have to do one multiplication $\mathbf{M}_{q^k}$ with the parabola. Carrying out the analysis the same way as in Section 9.3.1, one sees that it is possible to compute the parabola and the point $[2]R'+Q'$ with cost $(k/2)\mathbf{M}_q + 2\mathbf{I}_{q^{k/2}} + 6\mathbf{M}_{q^{k/w}} + 2\mathbf{S}_{q^{k/2}} +$

$2\mathbf{M}_{(\omega)} + \mathbf{M}_{(\omega^2)} + 5\mathbf{add}_{q^{k/2}} + 8\mathbf{sub}_{q^{k/2}} + \mathbf{add}_q + \mathbf{sub}_q$. Overall, the parabola technique saves $\mathbf{M}_{q^k} + (k/2)\mathbf{M}_{q^{k/2}} + \mathbf{S}_{q^{k/2}}$, clearly the **add** and **sub** costs are lower, but it uses two more multiplications by ω and ω^2, respectively. In this specific setting, the parabola method is indeed more efficient than the standard Miller loop with only doubling and addition steps. This effect becomes more pronounced for larger embedding degrees.

9.5 Squared pairings

For applications to cryptography, the idea of using the squared Weil or Tate pairing instead of the usual pairings was introduced in [28]. Peter was very fond of the idea of squared pairings. The denominator cancellation, which improves efficiency, was one very nice consequence, but that was also achieved independently in a different way in [8, 9]. However, an often unrecognized part of the history is that the squared Weil pairing may be viewed as a precursor to the definition of the ate pairing and optimal pairings which are in the same vein as the squared pairing. Namely, those more recent definitions consider a higher power of the Tate pairing and achieve greater efficiency by reducing the Miller loop length. The following exposition of the squared Weil and Tate pairings is a lightly edited version of Sections 2 and 3 from [28].

9.5.1 The squared Weil pairing

The squared pairing idea was introduced in [28] as a construction of new pairings, called the *squared Weil pairing* and *squared Tate pairing*. An analogous pairing for hyperelliptic curves was also introduced. In each case, a direct formula for evaluating the pairing was given, along with proofs that the values coincide with the value of the original pairing (Weil or Tate) squared. Here we present only the construction of the new pairing, and omit the proofs of correctness. These alternate pairings had the advantage of being more efficient to compute than Miller's algorithm for the original Weil and Tate pairings. The squared pairing also has the advantage that it is guaranteed to output the correct answer and does not depend on inputting a randomly chosen point. In contrast, Miller's algorithm may restart, since the randomly chosen point can cause the algorithm to fail.

Algorithm for $e_r(P, Q)^2$

Given two r-torsion points P and Q on E, we want to compute $e_r(P, Q)^2$. Start with an addition-subtraction chain for r. That is, after an initial 1, every element

in the chain is a sum or difference of two earlier elements, until an r appears. Well-known techniques give a chain of length $O(\log(r))$. For each j in the addition-subtraction chain, form a tuple $t_j = [[j]P, [j]Q, n_j, d_j]$ such that

$$\frac{n_j}{d_j} = \frac{f_{j,P}(Q)\, f_{j,Q}(-P)}{f_{j,P}(-Q)\, f_{j,Q}(P)}. \tag{9.7}$$

Start with $t_1 = [P, Q, 1, 1]$. Given t_j and t_k, the following procedure gets t_{j+k}.

1 Compute the points $[j]P + [k]P = [j + k]P$ and $[j]Q + [k]Q = [j + k]Q$.
2 Find coefficients of the line $l_{[j]P,[k]P}(X) = c_0 + c_1 x(X) + c_2 y(X)$.
3 Find coefficients of the line $l_{[j]Q,[k]Q}(X) = c'_0 + c'_1 x(X) + c'_2 y(X)$.
4 Set

$$n_{j+k} = n_j n_k (c_0 + c_1 x(Q) + c_2 y(Q))\,(c'_0 + c'_1 x(P) - c'_2 y(P)),$$
$$d_{j+k} = d_j d_k (c_0 + c_1 x(Q) - c_2 y(Q))\,(c'_0 + c'_1 x(P) + c'_2 y(P)).$$

A similar construction gives t_{j-k} from t_j and t_k. The vertical lines through $[j + k]P$ and $[j + k]Q$ do not appear in the formulas for n_{j+k} and d_{j+k}, because the contributions from Q and $-Q$ (or from P and $-P$) are equal. When $j + k = r$, this simplifies to $n_{j+k} = n_j n_k$ and $d_{j+k} = d_j d_k$, since c_2 and c'_2 will be zero.

When n_r and d_r are non-zero, then the computation

$$\frac{n_r}{d_r} = \frac{f_{r,P}(Q)\, f_{r,Q}(-P)}{f_{r,P}(-Q)\, f_{r,Q}(P)}$$

has been successful, and we have the correct output. If, however, n_r or d_r is zero, then some factor such as $c_0 + c_1 x(Q) + c_2 y(Q)$ must have vanished. That line was chosen to pass through $[j]P$, $[k]P$, and $-[j + k]P$, for some j and k. It does not vanish at any other point on the elliptic curve. Therefore this factor can vanish only if $Q = [j]P$ or $Q = [k]P$ or $Q = -[j + k]P$. In all of these cases Q will be a multiple of P, ensuring $e_r(P, Q) = 1$.

Overall, the squared Weil pairing advances from t_i and t_j to t_{i+j} with 12 field multiplications and 2 field divisions in the generic case, compared to 18 field multiplications and 2 field divisions for Miller's general method. When $i = j$, each algorithm needs 2 additional field multiplications due to the elliptic curve doublings.

9.5.2 The squared Tate pairing

Assume $P \in E(\mathbb{F}_q)[r]$, and $Q \in E(\mathbb{F}_{q^k})$, with neither being the identity and P not equal to a multiple of Q. Define

$$v_r(P, Q) := \left(\frac{f_{r,P}(Q)}{f_{r,P}(-Q)}\right)^{(q^k-1)/r},$$

where $f_{r,P}$ is as above, and call v_r the squared Tate pairing. To justify this terminology, it was shown in [28] that $v_r(P, Q) = t_r(P, Q)^2$.

Algorithm for $v_r(P, Q)$

Given an r-torsion point P on E and a point Q on E, we want to compute $v_r(P, Q)$. As before, start with an addition-subtraction chain for r. For each j in the addition-subtraction chain, form a tuple $t_j = [[j]P, n_j, d_j]$ such that

$$\frac{n_j}{d_j} = \frac{f_{j,P}(Q)}{f_{j,P}(-Q)}. \tag{9.8}$$

Start with $t_1 = [P, 1, 1]$. Given t_j and t_k, the following procedure gets t_{j+k}.

1 Compute the points $[j]P + [k]P = [j + k]P$.
2 Find the line function $g_{[j]P,[k]P}(X) = c_0 + c_1x(X) + c_2y(X)$.
3 Set

$$n_{j+k} = n_j \cdot n_k \cdot (c_0 + c_1x(Q) + c_2y(Q)),$$
$$d_{j+k} = d_j \cdot d_k \cdot (c_0 + c_1x(Q) - c_2y(Q)).$$

A similar construction gives t_{j-k} from t_j and t_k. The vertical lines through $[j + k]P$ and $[j + k]Q$ do not appear in the formulas for n_{j+k} and d_{j+k}, because the contributions from Q and $-Q$ are equal. When $j + k = r$, one can further simplify this to $n_{j+k} = n_j \cdot n_k$ and $d_{j+k} = d_j \cdot d_k$, since c_2 will be zero. When n_r and d_r are non-zero, then the computation of (9.8) with $j = r$ is successful, and after raising to the $(q^k - 1)/r$-th power, we have the correct output. If some n_r or d_r were zero, then some factor such as $c_0 + c_1x(Q) + c_2y(Q)$ must have vanished. That line was chosen to pass through $[j]P$, $[k]P$, and $-[j + k]P$, for some j and k. It does not vanish at any other point on the elliptic curve. Therefore this factor can vanish only if $Q = [j]P$ or $Q = [k]P$ or $Q = -[j + k]P$ for some j and k. In all of these cases Q would be a multiple of P, contrary to our assumption.

Bibliography

[1] D. F. Aranha, P. S. L. M. Barreto, P. Longa, and J. E. Ricardini. The realm of the pairings. In T. Lange, K. Lauter, and P. Lisonek, editors, *SAC 2013: 20th Annual International Workshop on Selected Areas in Cryptography*, volume 8282 of *Lecture Notes in Computer Science*, pages 3–25. Springer, Heidelberg, Aug. 2014. (Cited on pages 2 and 9.)

[2] D. F. Aranha, L. Fuentes-Castañeda, E. Knapp, A. Menezes, and F. Rodríguez-Henríquez. Implementing pairings at the 192-bit security level. In M. Abdalla and T. Lange, editors, *Pairing-Based Cryptography – Pairing 2012*, volume 7708 of *Lecture Notes in Computer Science*, pages 177–195. Springer, 2012. (Cited on page 2.)

[3] D. F. Aranha, K. Karabina, P. Longa, C. H. Gebotys, and J. López. Faster explicit formulas for computing pairings over ordinary curves. In K. G. Paterson, editor, *Advances in Cryptology – EUROCRYPT 2011*, volume 6632 of *Lecture Notes in Computer Science*, pages 48–68. Springer, Heidelberg, May 2011. (Cited on pages 2 and 9.)

[4] C. Arène, T. Lange, M. Naehrig, and C. Ritzenthaler. Faster computation of the Tate pairing. *Journal of Number Theory*, 131(5):842–857, 2011. (Cited on pages 16 and 17.)

[5] R. M. Avanzi, H. Cohen, C. Doche, G. Frey, T. Lange, K. Nguyen, and F. Vercauteren. *Handbook of Elliptic and Hyperelliptic Curve Cryptography*. Chapman & Hall/CRC Press, 2005. (Cited on page 30.)

[6] D. V. Bailey and C. Paar. Efficient arithmetic in finite field extensions with application in elliptic curve cryptography. *Journal of Cryptology*, 14(3):153–176, 2001. (Cited on page 10.)

[7] S. Baktir and B. Sunar. Optimal tower fields. *IEEE Transactions on Computers*, 53(10):1231–1243, 2004. (Cited on page 13.)

[8] P. S. L. M. Barreto, H. Y. Kim, B. Lynn, and M. Scott. Efficient algorithms for pairing-based cryptosystems. In M. Yung, editor, *Advances in Cryptology – CRYPTO 2002*, volume 2442 of *Lecture Notes in Computer Science*, pages 354–368. Springer, Heidelberg, Aug. 2002. (Cited on pages 2, 4, 7, and 26.)

[9] P. S. L. M. Barreto, B. Lynn, and M. Scott. Efficient implementation of pairing-based cryptosystems. *Journal of Cryptology*, 17(4):321–334, Sept. 2004. (Cited on pages 4, 7, and 26.)

[10] N. Benger and M. Scott. Constructing tower extensions of finite fields for implementation of pairing-based cryptography. In M. A. Hasan and T. Helleseth, editors, *Arithmetic of Finite Fields, Third International Workshop, WAIFI 2010, Istanbul, Turkey, June 27-30, 2010. Proceedings*, volume 6087 of *Lecture Notes in Computer Science*, pages 180–195. Springer, 2010. (Cited on page 10.)

[11] D. J. Bernstein and T. Lange. Explicit-Formulas Database, 2016. `https://hyperelliptic.org/EFD`. (Cited on pages 15 and 17.)

[12] 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. In M. Joye, A. Miyaji, and A. Otsuka, editors, *PAIRING 2010: 4th International Conference on Pairing-based*

Cryptography, volume 6487 of *Lecture Notes in Computer Science*, pages 21–39. Springer, Heidelberg, Dec. 2010. (Cited on pages 2 and 9.)

[13] I. F. Blake, G. Seroussi, and N. P. Smart, editors. *Elliptic Curves in Cryptography*. Cambridge University Press, 1999. (Cited on page 3.)

[14] I. F. Blake, G. Seroussi, and N. P. Smart, editors. *Advances in Elliptic Curve Cryptography*. Cambridge University Press, 2005. (Cited on page 31.)

[15] D. Boneh, G. Di Crescenzo, R. Ostrovsky, and G. Persiano. Public key encryption with keyword search. In C. Cachin and J. Camenisch, editors, *Advances in Cryptology – EUROCRYPT 2004*, volume 3027 of *Lecture Notes in Computer Science*, pages 506–522. Springer, Heidelberg, May 2004. (Cited on page 21.)

[16] D. Boneh and M. K. Franklin. Identity-based encryption from the Weil pairing. In J. Kilian, editor, *Advances in Cryptology – CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 213–229. Springer, Heidelberg, Aug. 2001. (Cited on page 1.)

[17] D. Boneh, E.-J. Goh, and K. Nissim. Evaluating 2-DNF formulas on ciphertexts. In J. Kilian, editor, *TCC 2005: 2nd Theory of Cryptography Conference*, volume 3378 of *Lecture Notes in Computer Science*, pages 325–341. Springer, Heidelberg, Feb. 2005. (Cited on page 1.)

[18] D. Boneh, B. Lynn, and H. Shacham. Short signatures from the Weil pairing. In C. Boyd, editor, *Advances in Cryptology – ASIACRYPT 2001*, volume 2248 of *Lecture Notes in Computer Science*, pages 514–532. Springer, Heidelberg, Dec. 2001. (Cited on page 1.)

[19] D. Boneh, B. Lynn, and H. Shacham. Short signatures from the Weil pairing. *Journal of Cryptology*, 17(4):297–319, Sept. 2004. (Cited on page 1.)

[20] D. Boneh, A. Sahai, and B. Waters. Functional encryption: Definitions and challenges. In Y. Ishai, editor, *TCC 2011: 8th Theory of Cryptography Conference*, volume 6597 of *Lecture Notes in Computer Science*, pages 253–273. Springer, Heidelberg, Mar. 2011. (Cited on page 1.)

[21] M. Ciet, M. Joye, K. Lauter, and P. L. Montgomery. Trading inversions for multiplications in elliptic curve cryptography. *Des. Codes Cryptography*, 39(2):189–206, 2006. (Cited on page 23.)

[22] C. Costello, H. Hisil, C. Boyd, J. M. González Nieto, and K. K.-H. Wong. Faster pairings on special Weierstrass curves. In H. Shacham and B. Waters, editors, *PAIRING 2009: 3rd International Conference on Pairing-based Cryptography*, volume 5671 of *Lecture Notes in Computer Science*, pages 89–101. Springer, Heidelberg, Aug. 2009. (Cited on pages 16 and 17.)

[23] C. Costello, T. Lange, and M. Naehrig. Faster pairing computations on curves with high-degree twists. In P. Q. Nguyen and D. Pointcheval, editors, *PKC 2010: 13th International Conference on Theory and Practice of Public Key Cryptography*, volume 6056 of *Lecture Notes in Computer Science*, pages 224–242. Springer, Heidelberg, May 2010. (Cited on pages 15, 16, and 17.)

[24] C. Doche. *Finite Field Arithmetic*, chapter 11 in [5], pages 201–237. CRC press, 2005. (Cited on pages 10 and 13.)

[25] S. Duquesne and G. Frey. *Background on Pairings*, chapter 6 in [5], pages 115–124. CRC press, 2005. (Cited on page 3.)

[26] S. Duquesne and G. Frey. *Implementation of Pairings*, chapter 16 in [5], pages 389–404. CRC press, 2005. (Cited on page 3.)

[27] K. Eisenträger, K. Lauter, and P. L. Montgomery. Fast elliptic curve arithmetic and improved Weil pairing evaluation. In M. Joye, editor, *Topics in Cryptology – CT-RSA 2003*, volume 2612 of *Lecture Notes in Computer Science*, pages 343–354. Springer, Heidelberg, Apr. 2003. (Cited on pages 1, 2, and 22.)

[28] K. Eisenträger, K. E. Lauter, and P. L. Montgomery. Improved Weil and Tate pairings for elliptic and hyperelliptic curves. In D. A. Buell, editor, *Algorithmic Number Theory, 6th International Symposium, ANTS-VI, Burlington, VT, USA, June 13-18, 2004, Proceedings*, volume 3076 of *Lecture Notes in Computer Science*, pages 169–183. Springer, 2004. (Cited on pages 1, 26, and 28.)

[29] D. Freeman, M. Scott, and E. Teske. A taxonomy of pairing-friendly elliptic curves. *Journal of Cryptology*, 23(2):224–280, Apr. 2010. (Cited on pages 8, 18, and 19.)

[30] 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. (Cited on page 8.)

[31] G. Frey and H.-G. Rück. A remark concerning m-divisibility and the discrete logarithm in the divisor class group of curves. *Mathematics of Computation*, 62(206):pp. 865–874, 1994. (Cited on pages 1 and 8.)

[32] S. D. Galbraith. *Pairings*, chapter IX in [14], pages 183–214. Cambridge University Press, 2005. (Cited on page 3.)

[33] S. D. Galbraith, K. Harrison, and D. Soldera. Implementing the Tate pairing. In C. Fieker and D. R. Kohel, editors, *Algorithmic Number Theory – ANTS*, volume 2369 of *Lecture Notes in Computer Science*, pages 324–337. Springer, 2002. (Cited on page 2.)

[34] P. Grabher, J. Großschädl, and D. Page. On software parallel implementation of cryptographic pairings. In R. M. Avanzi, L. Keliher, and F. Sica, editors, *SAC 2008: 15th Annual International Workshop on Selected Areas in Cryptography*, volume 5381 of *Lecture Notes in Computer Science*, pages 35–50. Springer, Heidelberg, Aug. 2009. (Cited on pages 20 and 21.)

[35] R. Granger and M. Scott. Faster squaring in the cyclotomic subgroup of sixth degree extensions. In P. Q. Nguyen and D. Pointcheval, editors, *PKC 2010: 13th International Conference on Theory and Practice of Public Key Cryptography*, volume 6056 of *Lecture Notes in Computer Science*, pages 209–223. Springer, Heidelberg, May 2010. (Cited on page 6.)

[36] R. Granger and N. Smart. On computing products of pairings. Cryptology ePrint Archive, Report 2006/172, 2006. `http://eprint.iacr.org/2006/172`. (Cited on pages 21 and 22.)

[37] J. Groth and A. Sahai. Efficient non-interactive proof systems for bilinear groups. In N. P. Smart, editor, *Advances in Cryptology – EUROCRYPT 2008*, volume 4965 of *Lecture Notes in Computer Science*, pages 415–432. Springer, Heidelberg, Apr. 2008. (Cited on page 21.)

[38] J. Guajardo and C. Paar. Itoh-Tsujii inversion in standard basis and its application in cryptography and codes. *Designs, Codes and Cryptography*, 25:207–216, 2001. (Cited on page 10.)

[39] J. E. Guzmán-Trampe, N. C. Cortés, L. J. D. Perez, D. O. Arroyo, and F. Rodríguez-Henríquez. Low-cost addition-subtraction sequences for the final

exponentiation in pairings. *Finite Fields and Their Applications*, 29:1–17, 2014. (Cited on page 6.)

[40] D. Hankerson, A. J. Menezes, and S. Vanstone. *Guide to Elliptic Curve Cryptography*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2004. (Cited on page 11.)

[41] F. Hess, N. P. Smart, and F. Vercauteren. The eta pairing revisited. *IEEE Transactions on Information Theory*, 52(10):4595–4602, 2006. (Cited on page 6.)

[42] S. Ionica and A. Joux. Another approach to pairing computation in Edwards coordinates. In D. R. Chowdhury, V. Rijmen, and A. Das, editors, *Progress in Cryptology - INDOCRYPT 2008: 9th International Conference in Cryptology in India*, volume 5365 of *Lecture Notes in Computer Science*, pages 400–413. Springer, Heidelberg, Dec. 2008. (Cited on pages 15 and 16.)

[43] T. Itoh and S. Tsujii. A fast algorithm for computing multiplicative inverses in GF(2^m) using normal bases. *Inf. Comput.*, 78(3):171–177, 1988. (Cited on page 10.)

[44] A. Joux. A one round protocol for tripartite diffie-hellman. In W. Bosma, editor, *Algorithmic Number Theory, 4th International Symposium, ANTS-IV, Leiden, The Netherlands, July 2-7, 2000, Proceedings*, volume 1838 of *Lecture Notes in Computer Science*, pages 385–394. Springer, 2000. (Cited on page 1.)

[45] A. Joux. A one round protocol for tripartite Diffie-Hellman. *Journal of Cryptology*, 17(4):263–276, Sept. 2004. (Cited on page 1.)

[46] T. Kobayashi, H. Morita, K. Kobayashi, and F. Hoshino. Fast elliptic curve algorithm combining Frobenius map and table reference to adapt to higher characteristic. In J. Stern, editor, *Advances in Cryptology – EUROCRYPT'99*, volume 1592 of *Lecture Notes in Computer Science*, pages 176–189. Springer, Heidelberg, May 1999. (Cited on pages 10 and 11.)

[47] N. Koblitz and A. Menezes. Pairing-based cryptography at high security levels (invited paper). In N. P. Smart, editor, *10th IMA International Conference on Cryptography and Coding*, volume 3796 of *Lecture Notes in Computer Science*, pages 13–36. Springer, Heidelberg, Dec. 2005. (Cited on page 10.)

[48] K. Lauter, P. L. Montgomery, and M. Naehrig. An analysis of affine coordinates for pairing computation. In M. Joye, A. Miyaji, and A. Otsuka, editors, *PAIRING 2010: 4th International Conference on Pairing-based Cryptography*, volume 6487 of *Lecture Notes in Computer Science*, pages 1–20. Springer, Heidelberg, Dec. 2010. (Cited on page 1.)

[49] A. Menezes, T. Okamoto, and S. A. Vanstone. Reducing elliptic curve logarithms to logarithms in a finite field. *IEEE Transactions on Information Theory*, 39(5):1639–1646, 1993. (Cited on pages 1 and 8.)

[50] V. S. Miller. The Weil pairing, and its efficient calculation. *Journal of Cryptology*, 17(4):235–261, Sept. 2004. (Cited on page 5.)

[51] B. Möller. Algorithms for multi-exponentiation. In S. Vaudenay and A. M. Youssef, editors, *SAC 2001: 8th Annual International Workshop on Selected Areas in Cryptography*, volume 2259 of *Lecture Notes in Computer Science*, pages 165–180. Springer, Heidelberg, Aug. 2001. (Cited on page 24.)

[52] P. L. Montgomery. Speeding the Pollard and elliptic curve methods of factorization. *Mathematics of Computation*, 48(177):243–264, 1987. (Cited on pages 2 and 13.)

[53] P. L. Montgomery. Five, six, and seven-term Karatsuba-like formulae. *IEEE Transactions on Computers*, 54(3):362–369, 2005. (Cited on page 11.)

[54] A. Sahai and B. R. Waters. Fuzzy identity-based encryption. In R. Cramer, editor, *Advances in Cryptology – EUROCRYPT 2005*, volume 3494 of *Lecture Notes in Computer Science*, pages 457–473. Springer, Heidelberg, May 2005. (Cited on page 1.)

[55] R. Sakai, K. Ohgishi, and M. Kasahara. Cryptosystems based on pairing. In *2000 Symposium on Cryptography and Information Security – SCIS 2000*, 2000. (Cited on page 1.)

[56] R. Schroeppel and C. Beaver. Accelerating elliptic curve calculations with the reciprocal sharing trick. Mathematics of Public-Key Cryptography (MPKC), University of Illinois at Chicago, 2003. (Cited on page 19.)

[57] M. Scott. Computing the Tate pairing. In A. Menezes, editor, *Topics in Cryptology – CT-RSA 2005*, volume 3376 of *Lecture Notes in Computer Science*, pages 293–304. Springer, Heidelberg, Feb. 2005. (Cited on pages 21 and 22.)

[58] M. Scott. On the efficient implementation of pairing-based protocols. In L. Chen, editor, *Cryptography and Coding – IMACC*, volume 7089 of *Lecture Notes in Computer Science*, pages 296–308. Springer, 2011. (Cited on page 21.)

[59] M. Scott, N. Benger, M. Charlemagne, L. J. D. Perez, and E. J. Kachisa. On the final exponentiation for calculating pairings on ordinary elliptic curves. In H. Shacham and B. Waters, editors, *Pairing-Based Cryptography - Pairing 2009, Third International Conference, Palo Alto, CA, USA, August 12-14, 2009, Proceedings*, volume 5671 of *Lecture Notes in Computer Science*, pages 78–88. Springer, 2009. (Cited on page 6.)

[60] J. H. Silverman. *The Arithmetic of Elliptic Curves*, volume 106 of *Graduate texts in mathematics*. Springer-Verlag, 1986. (Cited on page 3.)

[61] F. Vercauteren. Optimal pairings. *IEEE Transactions on Information Theory*, 56(1):455–461, 2010. (Cited on page 7.)

Subject index