Simpler and Faster Pairings from the Montgomery Ladder

Giacomo Pope[1,2], Krijn Reijnders[3], Damien Robert[4], Alessandro Sferlazza[5] and Benjamin Smith[a,6,7]

[1] NCC Group, United Kingdom
[2] University of Bristol, United Kingdom
[3] COSIC, KU Leuven, Belgium
[4] Inria Bordeaux, Institut de Mathématiques de Bordeaux, France
[5] Technical University of Munich, Germany
[6] Inria, France
[7] LIX, CNRS, École polytechnique, Institut Polytechnique de Paris, France

Abstract. We show that Montgomery ladders compute pairings as a by-product, and explain how a small adjustment to the ladder results in simple and efficient algorithms for the Weil and Tate pairing on elliptic curves using *cubical arithmetic*. We demonstrate the efficiency of the resulting *cubical pairings* in several applications from isogeny-based cryptography. Cubical pairings are simpler and more performant than pairings computed using Miller's algorithm: we get a speed-up of over 40% for use-cases in SQIsign, and a speed-up of about 7% for use-cases in CSIDH. While these results arise from a deep connection to biextensions and cubical arithmetic, in this article we keep things as concrete (and digestible) as possible. We provide a concise and complete introduction to cubical arithmetic as an appendix.

Keywords: pairings · isogenies · cubical arithmetic · Montgomery ladder

1 Introduction

Elliptic curves live in projective space almost by definition. Elliptic-curve cryptographers use projective coordinates not as a point of mathematical principle, but rather for their algorithmic advantages. The most obvious of these advantages is that projective coordinates can save us many expensive division operations: denominators in algebraic expressions can be multiplied into the projective Z-coordinate instead of divided out of the X and Y coordinates. Thus, arbitrarily many division-free projective group operations can be composed before the final normalization, where inverting the last Z-coordinate and multiplying through X and Y yields the desired result.

To make this more concrete: suppose we want to compute an x-only scalar multiplication operation on an elliptic curve $E/\mathbb{F}_q$. We are given the x-coordinate x_P of a point P on E and an integer m, and we want to compute the x-coordinate x_R of $R = [m]P$. The now-standard way to do this is the Montgomery ladder [Mon87; CS18]. The input is now

E-mail: giacomopope@gmail.com (Giacomo Pope), crypto.krijn@gmail.com (Krijn Reijnders), damien.robert@inria.fr (Damien Robert), alessandro.sferlazza@tum.de (Alessandro Sferlazza), smith@lix.polytechnique.fr (Benjamin Smith)

[a]This work was supported in part by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement ISOCRYPT - No. 101020788), by the Research Council KU Leuven grant C14/24/099 and by CyberSecurity Research Flanders with reference number VR20192203, by the HYPERFORM consortium, funded by France through Bpifrance, and by the France 2030 program under grant agreement ANR-22-PETQ-0008 PQ-TLS.

Date of this document: 2025-04-14.

a pair (X_P, Z_P) in $\mathbb{F}_q^2$ such that $x_P = X_P/Z_P$, together with the scalar m; the output is a pair (X_R, Z_R) in $\mathbb{F}_q^2$ such that $x_R = X_R/Z_R$. Only the ratio $(X_R : Z_R)$ (the projective point) matters for scalar multiplication—but the ladder does not output a ratio, it outputs a pair of field values. The distinction between the pair (X_R, Z_R) and the projective point $(X_R : Z_R)$ that it represents will be critical in this paper.

For most implementors, the projective factor Z_R is just an algebraic blob that must be finally divided out of X_R in order to recover the desired $x_R = X_R/Z_R$ for storage or transmission. Our aim is to convince them that the projective factors are not mere junk to be normalized away at the end of a computation: they have algebraically meaningful values, and those values turn out to be very closely related to the Tate pairing. Indeed, we will show that many useful pairings can be computed using projective factors from x-only scalar multiplications.

Our main tool is *cubical arithmetic* [Rob24], which uses simple x-only operations (very similar to Montgomery arithmetic) to compute the Weil and Tate pairings on Montgomery curves. The resulting algorithms are incredibly simple, yet still very efficient when compared to the traditional Miller-loop approach.

1.1 Contributions

We showcase the simplicity of cubical ladders as a tool for computing pairings by analyzing them in the context of three applications in isogeny-based cryptography.

- SQIsign [DKL+20; BDD+24; AAA+25] is based on supersingular curves $E/\mathbb{F}_{p^2}$, where p is of the form $p = 2^f \cdot g - 1$. It uses pairings of degree 2^f to compute change-of-basis matrices for the 2^f-torsion. This example showcases the simplicity of degree-2^f pairings, which mostly rely on doublings.
- SIKE [JAC+22] was based on supersingular curves $E/\mathbb{F}_{p^2}$ with $p = 2^{e_2} \cdot 3^{e_3} - 1$, and required pairings of degree 3^{e_3}. (Obviously, SIKE is broken [CD23; MMP+23; Rob23], but it remains a useful test of degree-3^k pairing performance.)
- CSIDH [CLM+18] uses supersingular curves $E/\mathbb{F}_p$ with $p = 2^f \cdot \prod \ell_i - 1$, where the ℓ_i are small odd primes; variants of CSIDH use pairings of degree $p+1$ or $\frac{p+1}{2^f}$. For these degrees, we show that cubical pairings are much simpler, and even more efficient, than pairings based on Miller loops.

To demonstrate the simplicity of cubical pairings for implementors, we provide proof-of-concept software packages in Sage and in Rust, targeting the isogeny-based applications mentioned above. These implementations are available at

https://github.com/GiacomoPope/cubical-pairings.

1.2 A guide to the paper

Section 2 recalls the basics of the Weil and Tate pairings, and Miller's algorithm [Mil04]. Section 3 presents the key motivating example: recovering the Tate pairing from projective factors after the Montgomery ladder. We present the elementary operations of cubical arithmetic—which is very close to Montgomery arithmetic—in Section 4, before using this to compute pairings in Section 5. We then consider implementation aspects in a variety of pairing use-cases from isogeny-based cryptography in Section 6. Our software implementations are described and benchmarked in Section 7.

Cubical arithmetic is not (yet) widely known in the cryptographic community, and our treatment in Section 4 gives only the bare minimum required to implement pairings. Appendix A provides a convenient overview of the deeper theory, and reproduces proofs of useful results. For readers who want even more detail, we recommend [Rob24].

Notation We work over $\mathbb{F}_q$, where q is a power of a prime $p > 3$. We write μ_n for the group of n-th roots of unity in $\overline{\mathbb{F}}_q$. The symmetric group on n elements is denoted $\mathfrak{S}_n$.

2 Preliminaries: pairings on elliptic curves

A pairing is a bilinear map $A \times B \to C$ between abelian groups A, B, and C. On elliptic curves, commonly used pairings are the Weil pairing [Wei40], and the Tate pairing [Tat62; Lic69; FR94] (sometimes called the Tate–Lichtenbaum pairing), which use subgroups or quotient groups of $E(\mathbb{F}_q)$ for A and B, and subgroups or quotient groups of $\mathbb{F}_q^*$ for C. These pairings are non-degenerate and Galois-invariant.

Pairings have countless constructive and destructive applications in discrete-logarithm-based cryptography, from the classic Menezes–Okamoto–Vanstone reduction [MVO91], Joux's one-round tripartite Diffie–Hellman [Jou00], Boneh and Franklin's IBE [BF01], and Boneh–Lynn–Shacham short signatures [BLS04], through to more recent applications in zero-knowledge proof systems [AHG23]. These applications generally involve cryptographic-sized DLP groups, and specialized fast pairing algorithms that require specially-parameterized curves and finite fields.

In this article, we are more concerned with *generic* pairings—that is, the Weil and Tate pairings on arbitrary curve groups where a high-speed cryptographic pairing, such as the Ate pairing [HSV06], is not available. This kind of pairing arises in isogeny-based cryptography [CJL+17; NR19; Rei23; CHM+23; MS24; Rob24], and also as a basic tool in algorithmic number theory.

2.1 The Weil Pairing

Fix an integer $n > 0$ coprime to the characteristic p of $\mathbb{F}_q$. The *Weil pairing* of degree n on an elliptic curve $E/\mathbb{F}_q$ is a non-degenerate, Galois-invariant pairing

$$e_{W,n} : E[n] \times E[n] \to \mu_n \,.$$

The Weil pairing is also *alternating*: $e_{W,n}(P, Q) = e_{W,n}(Q, P)^{-1}$ for all $P, Q \in E[n]$.

The Weil pairing can be used to determine linear independence of elliptic-curve points in order to compute the group structure (given the group order), and also to express points in terms of a given torsion basis. More concretely: fix a basis $\langle P, Q\rangle$ of $E[n]$ and let $\zeta := e_{W,n}(P, Q)$. Given $R \in E[n]$, we want to find $a, b \in \mathbb{Z}/n\mathbb{Z}$ such that $R = [a]P + [b]Q$. Using

$$e_{W,n}(P, R) = e_{W,n}(P, [a]P + [b]Q) = e_{W,n}(P, P)^a \cdot e_{W,n}(P, Q)^b = \zeta^b \,,$$

and

$$e_{W,n}(Q, R) = e_{W,n}(Q, [a]P + [b]Q) = e_{W,n}(Q, P)^a \cdot e_{W,n}(Q, Q)^b = \zeta^{-a} \,.$$

we can recover $b = \log_\zeta(e_{W,n}(P, R))$ and $a = -\log_\zeta(e_{W,n}(Q, R))$ by computing discrete logarithms in μ_n instead of in $E[n]$. This is always easier, if only because arithmetic in $\mathbb{F}_q$ is always more efficient than in $E/\mathbb{F}_q$, but also because in large instances, asymptotically faster discrete logarithm algorithms are available for finite fields.

2.2 The Tate Pairing

Suppose now $\mu_n \subset \mathbb{F}_q^*$. The Tate pairing of degree n is a non-degenerate, Galois-invariant pairing

$$e_{T,n} : E(\mathbb{F}_q)[n] \times E(\mathbb{F}_q)/nE(\mathbb{F}_q) \longrightarrow \mathbb{F}_q^*/(\mathbb{F}_q^*)^n, \,.$$

We emphasize that $e_{T,n}(P,Q)$ is only defined up to n-th powers in $\mathbb{F}_q$. The degree-n Tate pairing is deeply linked to divisibility by $[n]$, which explains its cryptographic applications in finding torsion bases. Other cryptographic applications often require unique representatives; we get these by exponentiation by $(q-1)/n$, which gives the *reduced Tate pairing*

$$e_{t,n} : E(\mathbb{F}_q)[n] \times E(\mathbb{F}_q)/nE(\mathbb{F}_q) \longrightarrow \mu_n\,, \quad (P,Q) \longmapsto e_{T,n}(P,Q)^{\frac{q-1}{n}}.$$

2.3 Computing pairings with Miller loops

The standard method for computing $e_{W,n}(P,Q)$ or $e_{T,n}(P,Q)$ is to evaluate *Miller functions* using the *Miller loop*. Given $P \in E$ and $n \in \mathbb{Z}$, the Miller function $f_{n,P}$ is any element of the function field $\mathbb{F}_q(E)$ with divisor

$$\operatorname{div}(f_{n,P}) = n(P) - ([n]P) - (n-1)(0_E).$$

If $P \in E[n]$, then $\operatorname{div}(f_{n,P}) = n(P) - n(0_E)$. Now, we can compute the Tate pairing as

$$e_{T,n}(P,Q) = f_{n,P}(D_Q),$$

where D_Q is any divisor linearly equivalent to $(Q) - (0_E)$ with support disjoint from $\{P, 0_E\}$. Similarly, the Weil pairing is computed as

$$e_{W,n}(P,Q) = f_{n,P}(D_Q)/f_{n,Q}(D_P).$$

When computing the *reduced* Tate pairing with embedding degree $k > 1$, we may evaluate $f_{n,P}$ at Q instead of D_Q: the difference disappears in the final exponentiation by $(q^k-1)/n$.

We can compute evaluations of Miller functions $f_{n,P}(Q)$ using Miller's algorithm [Mil04], which relies fundamentally on the relation

$$f_{(n+m),P} = f_{n,P} \cdot f_{m,P} \cdot l_{[n]P,[m]P}, \tag{1}$$

where $l_{[n]P,[m]P}$ is derived from the (geometric) lines that appear in the addition of $[n]P$ and $[m]P$ in the chord-and-tangent construction. Given $P \in E[n]$ and $Q \in E(\mathbb{F}_q)$, Miller's loop computes $f_{n,P}(Q)$ directly using a double-and-add approach: doubling is computed using Equation (1) with $n = m$, which gives

$$f_{2n,P}(Q) = f_{n,P}(Q)^2 \cdot l_{[n]P,[n]P}(Q)\,,$$

while addition takes $m = 1$, $f_{1,P} = 1$, which gives

$$f_{n+1,P}(Q) = f_{n,P}(Q) \cdot l_{[n]P,P}(Q)\,.$$

Optimizing Miller's loop to compute pairings as efficiently as possible is an active and specialized area of research.

Remark 1. Most elliptic scalar multiplication or isogeny computations can work on the Kummer line—using only x-coordinates—but most Miller-loop computations require general functions $f : E \to \mathbb{P}^1$ on the curve E. In contrast, cubical pairings (detailed below) operate entirely on the Kummer line. This explains the close resemblance to Montgomery arithmetic, and the efficiency compared to generic pairings using the Miller loop.

3 Tate pairings as a by-product of the ladder

We will now show how projective factors of Montgomery ladder outputs can be used to compute pairings. We will see that there is a certain correction factor involved, and removing this will motivate our use of cubical arithmetic in the following sections.

Let P be an n-torsion point on $E/\mathbb{F}_q$ with projective x-coordinate $(x_P : 1)$. If we compute $[n]P$ using the Montgomery ladder, then the result is a pair $(X_{[n]P}, Z_{[n]P})$ such that $(X_{[n]P} : Z_{[n]P}) = (1 : 0)$; in particular, $Z_{[n]P} = 0$, and the projective factor is $\lambda_P = X_{[n]P}$. Now let Q be any other point on E, with projective x-coordinate $(x_Q : 1)$. If we compute $[n]P + Q$ using the *three-point ladder* (a common variant of the Montgomery ladder [DJP14, Alg. 1] that computes $[n]P + Q$ given the coordinates of P, Q and their difference $P - Q$; see also Algorithm 3 below) then we obtain a pair $(X_{[n]P+Q}, Z_{[n]P+Q})$ such that $(X_{[n]P+Q} : Z_{[n]P+Q}) = (x_Q : 1)$; the projective factor is $\lambda_Q = Z_{[n]P+Q}$. These factors λ_P and λ_Q carry pairing information: the square of the Tate pairing $e_{T,n}(P, Q)^2$ is *almost* given by λ_Q/λ_P.

To get the correct values for the Tate pairing, λ_P and λ_Q need small adjustments. Let $l(k)$ denote the bit length of an integer k, and set $n_Q = 2^{l(n)} - n - 1$, $n_{P-Q} = n$, and $n_P = n_{P+Q} \cdot n_Q$. Then if we define

$$c_P := (4x_P)^{-n\cdot(2^{l(n)}-n)} \quad \text{and} \quad c_Q := (4x_P)^{-n_P} \cdot (4x_Q)^{-n_Q} \cdot (4x_{P+Q})^{-n_{P+Q}} ,$$

then

$$e_{T,n}(P, Q)^2 = \frac{c_Q}{c_P} \cdot \frac{\lambda_Q}{\lambda_P} .$$

Up to a small correction, the Montgomery ladder already computes pairings!

On closer inspection, the exponents in c_P and c_Q correspond to the number of differential additions xADD where X_P, X_Q, or X_{P+Q} is used as the difference point, where xADD : $x_P, x_Q, x_{P-Q} \mapsto x_{P+Q}$ is computed in projective coordinates as

$$\begin{aligned} X_{P+Q} &= Z_{P-Q}\left((X_P - Z_P)(X_Q + Z_Q) + (X_P + Z_P)(X_Q - Z_Q)\right)^2 , \\ Z_{P+Q} &= X_{P-Q}\left((X_P - Z_P)(X_Q + Z_Q) - (X_P + Z_P)(X_Q - Z_Q)\right)^2 . \end{aligned}$$

Hence, if we tweak this xADD operation by small factors to ensure that the resulting c_P and c_Q are trivial, then the projective factors λ_P and λ_Q really will compute (the square of) the Tate pairing. It suffices to multiply the formulæ for X_{P+Q} and Z_{P+Q} through by $1/(4X_{P-Q}Z_{P-Q})$ (remember that (X_{P-Q}, Z_{P-Q}) is constant throughout the Montgomery ladder). Doing this yields a function cADD : $(x_P, x_Q, x_{P-Q}) \mapsto x_{P+Q}$, given by

$$\begin{aligned} X_{P+Q} &= (4X_{P-Q})^{-1} \cdot \left((X_P - Z_P)(X_Q + Z_Q) + (X_P + Z_P)(X_Q - Z_Q)\right)^2, \\ Z_{P+Q} &= (4Z_{P-Q})^{-1} \cdot \left((X_P - Z_P)(X_Q + Z_Q) - (X_P + Z_P)(X_Q - Z_Q)\right)^2. \end{aligned}$$

This modified arithmetic has a deeper mathematical explanation, as it is an example of *cubical arithmetic* [Rob24] for the case of Montgomery curves. Cubical arithmetic uses the simple operations xDBL and the modified differential addition cADD to compute the Weil and Tate pairings on Montgomery curves in a way that is, compared to the traditional Miller loop, incredibly simple and yet very efficient: we simply compute $[n]P$ using cubical arithmetic to obtain λ_P, and $[n]P + Q$ to obtain λ_Q, and then $e_{T,n}(P, Q)^2 = \lambda_Q/\lambda_P$. We give a very quick overview of these cubical arithmetic operations in Section 4, before giving formulæ and algorithms for pairings in Section 5.

4 Cubical Arithmetic

As we saw above, slightly modifying the usual x-only differential addition on Montgomery curves allows us to easily compute the Weil and Tate pairings. The resulting arithmetic is called *cubical arithmetic*. In this section we present algorithms to compute Montgomery ladders and pairings with cubical arithmetic, We refer the interested reader to Appendix A for a brief exploration of the underlying theory, linking cubical arithmetic to pairing computations, and [Rob24, Sec. 5] for a more detailed discussion.

Remark 2. We stress that the correctness of cubical arithmetic crucially depends on the specific affine representation (X_P, Z_P) of the projective coordinate $(X_P : Z_P)$, contrary to ordinary x-only arithmetic.

4.1 Concrete algorithms for cubical arithmetic

Montgomery arithmetic is built on two fundamental functions: xDBL and xADD. The cubical analogues of xDBL and xADD are called cDBL and cADD, respectively. Helpfully, the doubling cDBL is identical to xDBL. The cubical differential addition cADD is slightly different to xADD, though the outputs are projectively equivalent. In fact, cADD requires the inverse coordinates of the difference point as input, and divides both coordinates by a factor of 4 at the end. (Despite the presence of these potentially expensive inversions, we will see in Section 5 that their impact on performance is minimal.)

When expressing costs, we consider all points and constants to be defined over a finite field $\mathbb{F}_q$. We denote multiplications by $\mathbf{M}$, squarings by $\mathbf{S}$, additions and subtractions by $\mathbf{A}$, division by 4 by $\mathbf{D_4}$, and multiplications by a curve constant by $\mathbf{M_c}$. In isogeny-based applications, we consider $\mathbf{M_c} = \mathbf{M}$.

Algorithm 1 Cubical doubling cDBL. Naturally constant-time.

Input: A cubical point $P = (X_P, Z_P)$ on a Montgomery curve E_A represented by $A_{24} = (A + 2)/4$.
Output: The cubical point $[2]P = (X_{[2]P}, Z_{[2]P})$.
Cost: $2\mathbf{M} + 1\mathbf{M_c} + 2\mathbf{S} + 4\mathbf{A}$

1: $t_0 \leftarrow (X_P + Z_P)^2$ ▷ 1S + 1A
2: $t_1 \leftarrow (X_P - Z_P)^2$ ▷ 1S + 1A
3: $X_{[2]P} \leftarrow t_0 \cdot t_1$ ▷ 1M
4: $t_2 \leftarrow t_0 - t_1$ ▷ 1A
5: $t_0 \leftarrow A_{24} \cdot t_2$ ▷ 1Mc
6: $Z_{[2]P} \leftarrow t_2 \cdot (t_0 + t_1)$ ▷ 1M + 1A
7: **return** $[2]P = (X_{[2]P}, Z_{[2]P})$

Algorithm 2 Cubical differential addition cADD. Naturally constant-time.

Input: Cubical points $P = (X_P, Z_P)$, $Q = (X_Q, Z_Q)$, and the inverse coordinates $\text{inv}(P - Q) = (X_{P-Q}^{-1}, Z_{P-Q}^{-1})$.
Output: The cubical point $P + Q = (X_{P+Q}, Z_{P+Q})$.
Cost: $4\mathbf{M} + 2\mathbf{S} + 6\mathbf{A} + 2\mathbf{D_4}$

1: $t_0 \leftarrow (X_P - Z_P) \cdot (X_Q + Z_Q)$ ▷ 1M + 2A
2: $t_1 \leftarrow (X_P + Z_P) \cdot (X_Q - Z_Q)$ ▷ 1M + 2A
3: $X_{P+Q} \leftarrow X_{P-Q}^{-1} \cdot (t_0 + t_1)^2$ ▷ 1M + 1S + 1A
4: $Z_{P+Q} \leftarrow Z_{P-Q}^{-1} \cdot (t_0 - t_1)^2$ ▷ 1M + 1S + 1A
5: $(X_{P+Q}, Z_{P+Q}) \leftarrow (X_{P+Q}/4, Z_{P+Q}/4)$ ▷ 2D4
6: **return** $P + Q = (X_{P+Q}, Z_{P+Q})$

Remark 3. Combining cDBL and cADD in a single cDBLADD routine, re-using shared values, decreases the total cost by $2\mathbf{A}$.

The cubical Montgomery ladder is the usual ladder, with cDBL and cADD in place of xDBL and xADD, respectively. Algorithm 3 is the cubical *three-point ladder*, which we need for pairing computations.

Algorithm 3 Cubical three-point cLADDER. Naturally constant-time.

Input: A b-bit integer $n = \sum_{i=0}^{b-1} n_i 2^i$ and cubical points $P, Q, P-Q$ on a Montgomery curve E_A, represented as $\big(R = (X_R, Z_R), \mathrm{inv}(R) = (X_R^{-1}, Z_R^{-1})\big)$.
Output: Cubical points $[n]P = (X_{[n]P}, Z_{[n]P})$, $[n]P + Q = (X_{[n]P+Q}, Z_{[n]P+Q})$
Cost: $b \cdot \mathsf{cDBL} + 2b \cdot \mathsf{cADD}$

$(S_0, S_1, T) \leftarrow (0, P, Q)$ ▷ Initialize the ladder
for $i = b-1$ down to 0 **do** ▷ set $n_b = 0$
  $R \leftarrow \mathsf{cADD}(S_0, S_1; \mathrm{inv}(P))$
  $\mathsf{SWAP}(S_0, S_1; n_i \oplus n_{i+1})$
  $\mathsf{SWAP}(\mathrm{inv}(Q), \mathrm{inv}(P-Q); n_i \oplus n_{i+1})$
  $T \leftarrow \mathsf{cADD}(T, S_0; \mathrm{inv}(Q))$
  $S_0 \leftarrow \mathsf{cDBL}(S_0; E_A)$
  $S_1 \leftarrow R$
end for
return S_0, T ▷ $S_0 = [n]P$ and $T_0 = [n]P + Q$

5 Cubical Pairings

We now describe how to use cubical ladders to compute the (non-reduced) Tate and Weil pairings of degree $\ell \geq 2$. There are slight differences for odd and even ℓ, which we describe in Sections 5.1 and 5.2 respectively. The key formulæ are derived and proven in [Rob24, Theorem 4.19], and we restate them here without proof (though the essentials of the proof are reproduced for completeness and easy reference in Theorem 3 in Appendix A.6 and Theorem 5 in Appendix A.10). We give complete algorithms for the Weil and Tate pairings in Section 5.3, then discuss optimizations and generalizations in Sections 5.4 and 5.5.

5.1 Odd-degree pairings

Cubical arithmetic gives us a straightforward way to compute the square of the Tate and Weil pairings. When ℓ is odd, the square of a pairing contains as much arithmetical information as the pairing itself, as we can recover one from the other via an exponentiation.

Let $P \in E(\mathbb{F}_q)[\ell]$ and $Q \in E(\mathbb{F}_q)$, then the square of the ℓ-Tate pairing is

$$e_{T,\ell}(P, Q)^2 = \frac{Z_{[\ell]P+Q}}{Z_Q \cdot X_{[\ell]P}},$$

up to ℓ-th powers in $\mathbb{F}_q$, where $[\ell]P$ and $[\ell]P + Q$ are computed from $P, Q, P-Q$ using cubical ladders. Similarly, given $P, Q \in E[\ell]$, the square of the Weil pairing can be computed as the ratio of two squared Tate pairings:

$$e_{W,\ell}(P, Q)^2 = \frac{Z_{[\ell]P+Q}}{Z_Q \cdot X_{[\ell]P}} \cdot \frac{Z_P \cdot X_{[\ell]Q}}{Z_{P+[\ell]Q}}.$$

5.2 Even-degree pairings

When $\ell = 2n$ is even, the square of the pairing has one bit less of information than the pairing. However, we can recover the full pairing value by replacing the ladders for ℓ with ladders for n, and using the *affine translation* by 2-torsion points given in Algorithm 4.

Given $P \in E(\mathbb{F}_q)[\ell]$ and $Q \in E(\mathbb{F}_q)$, set $P_0 = [n]P$. We first compute $[n]P$, $[n]P + Q$ from $P, Q, P-Q$ using cLADDER, then we apply an affine translation by P_0 to the output. The resulting expression for the Tate pairing is

$$e_{T,\ell}(P, Q) = \frac{Z_{[n]P+Q+P_0}}{Z_Q \cdot X_{[n]P+P_0}} \cdot \left(\frac{Z_P \cdot Z_Q}{Z_{P+Q}}\right)^n. \tag{2}$$

If we compute the Weil pairing as the ratio of two Tate pairings, then the n-th powers from Equation (2) cancel out. We can also avoid this exponentiation in the Tate pairing if we normalize the points $P, Q, P-Q$ to have Z-coordinate 1. Indeed, $Z_P = Z_Q = 1$, and the value Z_{P+Q} output by $\mathsf{cADD}(P, Q, P-Q)$ is a square.

Algorithm 4 Affine translation translate. Can be made constant-time with swaps.

Input: A cubical point $P = (X_P, Z_P)$ on a Montgomery curve E_A and a cubical point $T = (r, s)$ in $E_A[2]$.
Output: The affine translation $Q = P + T$.
Cost: $4\mathbf{M} + 2\mathbf{A}$

$X_Q \leftarrow r \cdot X_P - s \cdot Z_P$ ▷ 2M + 1a
$Z_Q \leftarrow s \cdot X_P - r \cdot Z_P$ ▷ 2M + 1a
if $s = 0$ **then**
    $(X_Q, Z_Q) \leftarrow (X_P, Z_P)$ ▷ Use constant-time swap
else if $r = 0$ **then**
    $(X_Q, Z_Q) \leftarrow (Z_P, X_P)$ ▷ Use constant-time swap
end if
return $Q = (X_Q, Z_Q)$

5.3 Concrete cubical pairing algorithms

Algorithms 5 and 6 give concrete algorithms to compute the Tate and Weil pairings, respectively, for any degree ℓ, assuming normalized input points P, Q and $P-Q$.

Algorithm 5 Tate pairing

Input: An integer ℓ, a Montgomery curve E_A, and normalized points $P, Q, P-Q$ represented as $\big(R = (x_R, 1), \mathrm{inv}(R) = (x_R^{-1}, 1)\big)$, with $P \in E_A(\mathbb{F}_q)[\ell]$, $Q \in E_A(\mathbb{F}_q)$.
Output: The non-reduced Tate pairing $e_{T,\ell}(P, Q)$ for even ℓ, or $e_{T,\ell}(P, Q)^2$ for odd ℓ

if ℓ is even **then** $n \leftarrow \ell/2$ **else** $n \leftarrow \ell$
$[n]P,\ [n]P + Q \leftarrow \mathsf{cLADDER}(n, E_A; P, Q, P-Q)$
if ℓ is even **then**
    $[\ell]P + Q \leftarrow \mathsf{translate}([n]P + Q, [n]P)$
    $[\ell]P \leftarrow \mathsf{translate}([n]P, [n]P)$
end if
$\lambda_Q \leftarrow Z_{[\ell]P+Q}$
$\lambda_P \leftarrow X_{[\ell]P}$
return λ_Q/λ_P

Algorithm 6 Weil pairing

Input: An integer ℓ, a Montgomery curve E_A, and points $P, Q, P-Q$ represented as in Tate, with $P, Q \in E[\ell]$.
Output: The Weil pairing $e_{W,\ell}(P, Q)$ for even ℓ, or $e_{W,\ell}(P, Q)^2$ for odd ℓ

$t_1 \leftarrow \mathsf{Tate}(\ell, E_A; P, Q, P-Q)$
$t_2 \leftarrow \mathsf{Tate}(\ell, E_A; Q, P, P-Q)$
return t_1/t_2

5.4 Optimizations for pairing computations

Our pairing algorithms admit several general optimizations, which we discuss in this section. (Application-specific optimizations are described in Section 6.)

5.4.1 Minimizing inversions

cADD requires the inverse $\mathrm{inv}(P-Q)$ of the difference point $P-Q$. Consequently, cLADDER requires inverses whenever cADD is used, which is twice per bit. But we can save almost all of these inversions:

1. *Normalized inputs.* Normalizing difference points as $R=(x_R,1)$ saves a multiplication by Z_R^{-1} in Line 4 of cADD.

2. *Precomputing difference points.* cLADDER only uses cADD with difference points $R\in\{P,Q,P-Q\}$. We therefore only need to invert x_P, x_Q and x_{P-Q} once, before the ladder, to use $\mathrm{inv}(R)=(x_R^{-1},1)$ as input to cADD.

3. *Combined inversion in pre-computation.* We can precompute the inverses of x_P, x_Q, and x_{P-Q} at the cost of one initial batch inversion.

We can also combine Steps (1) and (3), batch-inverting X_R and Z_R to get x_R and x_R^{-1} with a single inversion. Together, this reduces the number of inversions in the complete ladder to just one.

5.4.2 Subfield eliminations

When computing the reduced Tate pairing, the final exponentiation of $\zeta = e_{T,\ell}(P,Q)$ by $(q-1)/\ell$ clears all factors in $\mathbb{F}_{p^a}$ for $p^a-1 \mid \frac{q-1}{\ell}$. This adds significant flexibility: we may introduce or remove computations that only affect ζ up to $\mathbb{F}_{p^a}^*$-factors. In most applications in isogeny-based cryptography, we have $q=p^2$ and $p-1 \mid (q-1)/\ell$, so we can introduce or remove factors in $\mathbb{F}_p^*$:

1. *Avoiding division by 4.* We can remove Line 5 of Algorithm 2 when computing the reduced Tate and Weil pairings. This does not change the output of cADD as a projective point, but it scales the output of cLADDER by some power of 4. In the reduced Tate pairing, this power of 4 is cleared by the final exponentiation; in the Weil pairing, it cancels out in the quotient of the two non-reduced Tate pairings.

2. *Ignoring subfield-rational points.* If $x_P \in \mathbb{F}_{p^a}$, then $[\ell]P=(X_{[\ell]P},0)$ is also $\mathbb{F}_{p^a}$-rational.[1] Thus, we can skip the final division by $X_{[\ell]P}$ in Line 9 of the Tate pairing.

3. *Relaxed offset.* Similarly, if $x_Q \in \mathbb{F}_{p^a}$, then $Z_{[n]Q} \in \mathbb{F}_{p^a}$. Whenever ℓ and n are coprime, we may use $Z_{[\ell]P+[n]Q}$ instead of $Z_{[\ell]P+Q}$, and ignore Z_Q, in Line 9 of the Tate pairing; this effectively computes $e_{T,\ell}(P,[n]Q)=e_{T,\ell}(P,Q)^n$. We can recover the correct value of $e_{T,\ell}(P,Q)$, if necessary, by adjusting the final exponentiation.[2]

5.4.3 Multiple pairings with the same point.

Some applications require the computation of $e_\ell(P,Q_i)$ for $i=1,\dots,m$. The computation of m parallel ladders with different offsets Q_i can be batched together, bringing the collective cost of these ladders down to $1\cdot\mathsf{cDBL}+m\cdot\mathsf{cADD}$ per bit.

[1] Note however that x_{P+Q} need not be $\mathbb{F}_{p^a}$-rational, even when x_P and x_Q both are.
[2] Some applications only need the order of $e_{T,\ell}(P,Q)$, which is unaffected by coprime exponents.

5.5 Beyond the ladder

The cubical ladder is easy to implement, but uses fixed base points for differential additions, as varying the base point requires an expensive inversion every time. Two observations allow for more flexibility.

1. *Inversion by conjugation.* If $x_R \in \mathbb{F}_{p^2} \setminus \mathbb{F}_p$, then $x_R^{-1} = \overline{x_R}/N(x_R)$, where $\overline{x_R}$ denotes the conjugate and $N(x_R) \in \mathbb{F}_p$ the norm of x_R. Multiplying X_{P-Q} resp. Z_{P-Q} by $\overline{X_{P-Q}} \cdot N(Z_{P-Q})$ resp. $\overline{Z_{P-Q}} \cdot N(X_{P-Q})$ in cADD, does not change the projective point, and the extra projective factor in cLADDER output is cleared by final exponentiation. This allows us to compute $[\ell]P + Q$ using cADD with varying base points at an extra cost of $2\mathbf{M}$ and two norm computations per cADD. In theory, this should reduce the cost of differential addition chains with variable differences. However, we were unable to find an approach that outperforms the standard cubical ladder, due to the increased cost per cADD.

2. *Double-and-Add.* As described in [Rob24], we may also use a double-and-add approach to compute $[\ell]P + Q$ and $[\ell]P$. The standard cubical ladder keeps track of the cubical points $[u]P+Q, [u]P, [u+1]P$; double-and-add forgets about $[u+1]P$. For a doubling step, we can compute $[2u]P + Q, [2u]P$ without $[u + 1]P$; this only costs one cDBL and one cADD. For a double-and-add step, we can reconstruct $[u + 1]P$ on the fly, *up to an unknown scalar*, using a (generally costly) *compatible addition* $[u + 1]P = ([u]P) + (P) = ([u]P + Q) + (P - Q)$ (see [LR16]). We then compute $[2u+1]P+Q$ and $[2u+1]P$ via two cADDs, both with $[u+1]P$; the unknown scalar disappears in the pairing formula.[3]

   This approach allows improvements such as non-adjacent forms (NAFs) and windowing methods, but it also increases implementation complexity, and makes constant-time implementation difficult. These optimized approaches may outperform the standard cubical ladder, e.g., when the Hamming weight of ℓ is high or low, but this is beyond the scope of this work as it requires an in-depth performance analysis of a specific situation. Such an analysis is often worthwhile whenever pairing performance is critical to an application. In isogeny-based cryptography, however, pairing computations are usually only a small part of a larger computational effort. We will see in Section 6 that the simple cubical ladder is already fast enough for our purposes.

5.6 Generalizations

Beyond the Weil and Tate pairing, we can also compute other pairings, such as the (optimal) Ate pairing, using cubical arithmetic [Rob24]. We emphasize that cubical arithmetic is not restricted to curves in Montgomery models: a cubical ladder can be defined for the Kummer line of any curve, once proper differential addition and doubling formulæ have been derived. Even more generally, pairings can be computed using cubical arithmetic on (Kummer varieties of) abelian varieties [LR16; Rob24]. This is applied in e.g. [CR24] to analyze the practical efficiency of cubical arithmetic for 2-Tate pairings. A practical Sage implementation of pairings on abelian surfaces can be found at [Sfe24].

6 Applications in Isogeny-based Cryptography

To showcase the simplicity, efficiency, and flexibility of cubical pairings, we consider three applications of pairings in isogeny-based cryptography. The field primes, elliptic curves,

[3] On a more technical note, what happens here is that we compute pairings through monodromy information, for which biextensions are the correct geometrical object. The cubical arithmetic presented here is a refinement of arithmetic on these biextensions: we can somewhat relax the cubical arithmetic to compute pairings, as long as we still get the correct biextension arithmetic.

point orders, and pairing degrees involved in isogeny-based cryptography are generally not compatible with specialized fast pairings, so our general cubical pairings can make real improvements. The three applications here rely on the same observation: arithmetic in $\mathbb{F}_q^*$ is much faster than arithmetic on $E(\mathbb{F}_q)$, so it is often worthwhile to compute a relatively costly pairing to move a calculation into $\mathbb{F}_q^*$ (and even more so when the pairing maps into μ_{p+1}, which boasts even faster arithmetic [Sta03]).

6.1 SQIsign: Change of Basis

Recent variants [BDD+24; AAA+25] of SQIsign [DKL+20] work with field primes in the form $p = 2^f \cdot g - 1$, where g is a small cofactor, and use pairings to compute the change of basis matrix M that maps a deterministic basis P, Q of $E[2^e]$ with $e \leq f$ into another basis R, S by

$$\begin{pmatrix} x_1 & x_2 \\ x_3 & x_4 \end{pmatrix} \cdot \begin{pmatrix} P \\ Q \end{pmatrix} = \begin{pmatrix} R \\ S \end{pmatrix},$$

The coefficients $x_1, x_2, x_3, x_4 \in \mathbb{Z}/2^e\mathbb{Z}$ can be computed with a few discrete logarithms.

Use case. We differentiate three cases:

1. When $e < f$ and both (P, Q) and (R, S) are a basis for $E[2^e]$, we compute five degree-2^e Weil pairings:

$$\zeta_0 = e_{W,2^e}(P, Q), \quad \zeta_1 = e_{W,2^e}(Q, R)^{-1}, \quad \zeta_2 = e_{W,2^e}(P, R),$$
$$\zeta_3 = e_{W,2^e}(Q, S)^{-1}, \text{ and } \zeta_4 = e_{W,2^e}(P, S),$$

   then solve the four discrete logarithms $x_i = \log_{2^e}(\zeta_0, \zeta_i)$ in $\mathbb{Z}/2^e\mathbb{Z}$.

2. When $e = f$, we can replace the Weil pairings in Case 1 by with degree-2^f Tate pairings.

3. When $e < f$ and $\langle P, Q\rangle = E[2^f]$ while $\langle R, S\rangle = E[2^e]$ (or vice versa), we can replace the Weil pairings in Case 1 with Tate pairings as long as we compute the Tate pairing with respect to the basis (R, S) in the first position, and the discrete logarithms with respect to $e_{T,2^f}(P, Q)$.

In practice, almost all changes of basis have either (P, Q) or (R, S) derived deterministically as a basis for $E[2^f]$, which resolves to Case 2 or Case 3. This is preferred over Case 1, as Tate pairings are faster to compute than Weil pairings.

Remark 4. In some cases, we are given a basis (R, S) of $E[2^e]$ and (P, Q) of $E[2^f]$, and want to know the coefficients $x_i \in \mathbb{Z}_{2^e}$ such that $P' = x_1R + x_2S$, $Q' = x_3R + x_4S$, where $P' = [2^{f-e}]P$ and $Q' = [2^{f-e}]Q$. This resolves to Case 3, by inverting the matrix M obtained from expressing (R, S) in terms of (P, Q).

Computation. Two optimizations are possible in this use case. First, we are computing multiple pairings with the same point twice allowing us to reuse several values. Second, pairings of degree 2^e only require, per bit k, one cDBL to compute $[2^k]P$, given $[2^{k-1}]P$ and A, and one cADD to compute $Q + [2^k]P$, given $Q + [2^{k-1}]P$ and $2[^{k-1}]P$, with fixed base point Q. This makes such pairings highly efficient.

Performance. We compare the performance of cubical pairings versus those using the Miller loop as optimized in [CLZ24] to compute a change-of-basis matrix M for the three primes used in SQIsign: $p_1 = 2^{248} \cdot 5 - 1$, $p_3 = 2^{376} \cdot 65 - 1$, and $p_5 = 2^{500} \cdot 27 - 1$ [BDD+24; AAA+25]. Table 1 compares the cubical and Miller approaches for computing pairings of degree 2^e with $e = \lfloor f/2 \rfloor$ in Case 3, .

Table 1: Operation counts (in $\mathbb{F}_{p^2}$-operations) for change of basis using degree-$2^{\lfloor f/2 \rfloor}$ pairings. Cost model: $\mathbf{M} = 1$, $\mathbf{S} = 0.8$, $\mathbf{A} = 0.15$; inversion $= 47\mathbf{M}$, square root $= 222\mathbf{M}$.

	Cubical Ladder (this work)				Miller Loop [CLZ24]				Gain
	M	**S**	**A**	**Total**	**M**	**S**	**A**	**Total**	
p_1	4 319	3 115	6 458	**7780**	8 242	4 968	10 046	**13723**	43.3%
p_3	6 259	4 735	9 786	**11515**	12 230	7 548	15 230	**20553**	44.0%
p_5	8 109	6 275	13 010	**15080**	16 064	10 018	20 252	**27116**	44.4%

6.2 SIKE: Public Key Compression

SIKE [JAC+22] used primes of the shape $p = 2^a \cdot 3^b - 1$, and pairings of degree 2^a and 3^b to compress public keys. These public keys can now be broken in polynomial time, of course, but their compression is still a useful test-case for pairings of degree 3^b.

Use case. In general, a basis (R, S) for $E[\ell^k]$ can be encoded as three affine x-coordinates (x_R, x_S, x_{R-S}) with $x_i \in \mathbb{F}_q$ in $3 \log q$ bits. In SIKE [JAC+22], public keys consist of such a basis for $\ell = 2$ or $\ell = 3$. To save space, we can compress such a representation (x_R, x_S, x_{R-S}): we compute a deterministic basis (P, Q) for $E[\ell^k]$, and express R and S in terms of P and Q, as done in Section 6.1 for $\ell = 2$. We can then represent the basis (R, S) by the values $x_i \in \mathbb{Z}/\ell^k\mathbb{Z}$ such that $R = [x_1]P + [x_2]Q$ and $S = [x_3]P + [x_4]Q$. This requires Weil or Tate pairings of degree ℓ^k. We already analyzed the performance of degree-2^k pairings in Section 6.1, so we focus on degree-3^b pairings in this section.

Computation. We can derive cubical tripling formulæ (closely matching Montgomery tripling formulæ [JAC+22]) like those used in Miller loop computations, but the practical improvement in deriving $Q + [3]P$ given Q and P seems marginal: cubical tripling costs about the same as a combined cDBL and cADD. In practice, we find that the standard cubical ladder cLADDER performs on par or better than variants using tripling formulæ.

Performance. We compare the performance of cubical pairings using cLADDER to pairings computed using the Miller loop with specialized tripling formulæ [CLZ24]. Table 2 compares the cost of 3^b-pairing computations for five SIKE parameter sets.

Table 2: Operation counts for degree-3^b pairings for the five SIKE parameter sets. Cost model: $\mathbf{M} = 1$, $\mathbf{S} = 0.8$, $\mathbf{A} = 0.15$, inversion $= 47\mathbf{M}$, square root $= 222\mathbf{M}$.

	Cubical Ladder (this work)				Miller Loop [CLZ24]				Gain
	M	**S**	**A**	**Total**	**M**	**S**	**A**	**Total**	
p_{434}	8 917	5 892	13 132	**15600**	18 277	6 276	23 621	**26841**	41.9%
p_{503}	10 242	6 832	15 232	**17992**	21 131	7 282	27 427	**31071**	42.1%
p_{610}	12 233	8 251	18 352	**21587**	25 432	8 811	33 136	**37451**	42.3%
p_{751}	15 010	10 214	22 792	**26600**	31 501	10 932	41 267	**46437**	42.7%
p_{964}	18 847	12 962	28 732	**33526**	39 635	13 858	51 993	**58520**	42.7%

6.3 CSIDH: Public Key Verification

CSIDH [CLM+18] works with field primes of the form $p = 2^f \cdot \prod \ell_i - 1$, where the ℓ_i are small odd primes. [Rei23] and [CLZ24] use Miller-loop-based pairings of degree $p + 1$

or $(p+1)/2^f$ to speed up several subroutines in CSIDH [CLM+18] and its deterministic variants dCSIDH/dCTIDH [CCC+24; CHMR25], mostly related to public-key verification. We will see that cubical pairings bring further speedups.

Use case.

1. For CSIDH, we must verify the supersingularity of the public-key curve E_A. We sample two random points P and Q, multiply both by an appropriate cofactor, and compute $\zeta = e_{T,r}(P,Q)$ using a cubical pairing. If the order of ζ is larger than $4\sqrt{p}$, then E_A is supersingular.

2. For dCSIDH/dCTIDH, the public keys have the form (E_A, u), where u is some seed to be expanded into two points $P \in E_A(\mathbb{F}_p)$, $Q \in E_A^t(\mathbb{F}_p)$ (here E_A^t is the quadratic twist). We must verify that both P and Q have order divisible by $L = \prod \ell_i$ (this also implicitly verifies the supersingularity of E_A). Multiply both points by $\frac{p+1}{L}$, and compute $\zeta = e_{T,L}(P,Q)$. The order of ζ is exactly L precisely when L divides the order of both P and Q.

We can use Tate pairings for both of the cases above, as described in Algorithms 4 and 5 from [Rei23]. We focus only on the performance of the pairings, as both the cofactor multiplication and the verification of the order of ζ are independent of the choice of pairing algorithm.

Computation. In contrast to Sections 6.1 and 6.2, these pairings have almost-primorial degree $L = \prod \ell_i$. As a result, we have to use the standard cubical ladder cLADDER instead of only doublings and triplings. However, since $x_P, x_Q \in \mathbb{F}_p$, many operations take place in $\mathbb{F}_p$ instead of $\mathbb{F}_{p^2}$. We therefore adjust the standard cubical ladder to work over $\mathbb{F}_p$ whenever a cDBL or cADD depends only on (a multiple of) P or Q.

Performance. We compare the performance of cubical pairings and Miller-loop pairings as optimized in [CLZ24]. Table 3 gives results for Case 1, using the CSIDH-512 prime $p = 4 \cdot \prod_{i=1}^{74} \ell_i - 1$, where $\ell_1, \ldots, \ell_{73}$ are the first 73 odd primes and $\ell_{74} = 587$. Table 4 gives results for Case 2, using the 2048-bit prime p_{194} from [CHMR25], and computing pairings of degree $r = \prod_{i=1}^{194} \ell_i$, where $\ell_1, \ldots, \ell_{194}$ are the first 194 odd primes.

Table 3: Operation counts for pairing-based supersingularity verification in CSIDH-512 (Case 1), averaged over 1000 runs. Cost model: $\mathbf{M} = 1$, $\mathbf{S} = 0.67$, $\mathbf{A} = 0.08$, with $\mathbb{F}_{p^2}$-multiplication $= 3\mathbf{M}$, $\mathbb{F}_{p^2}$-squaring $= 2\mathbf{M}$, inversion $= 64\mathbf{M}$, square root $= 459\mathbf{M}$.

Method	**M**	**S**	**A**	**Total**
Alg. 5, cubical (this work)	8 267	3 687	11 076	**11623**
Alg. 5, Miller [CLZ24]	8 050	4 592	11 041	**12011**
Doliskani's [BGS22]	13 803	0	10 228	**14621**
Doliskani's [Rob24]	13 352	2	10 220	**14171**

Remark 5. Algorithm 5 of [Rei23] has a small chance to require a repetition, depending on the choice of N, when not enough torsion is collected. More precisely, if $\zeta < 4\sqrt{p}$, it is most likely that E is supersingular, but the random points P and Q were simply missing some torsion, usually only one or two ℓ_i. We can verify the supersingularity by taking a random new point R and see if this has the missing torsion ℓ_i. The additional cost is roughly a scalar multiplication of length $\log p$ bits. Doliskani's algorithm [Dol18] performs most operations over $\mathbb{F}_{p^2}$. Note that the probabilistic supersingularity test that checks

Table 4: Operation counts for pairing-based dCTIDH-2048 public-key verification (Case 2). Cost model: $\mathbf{M} = \mathbf{S} = 1$, $\mathbf{A} = 0$, with $\mathbb{F}_{p^2}$-multiplication $= 3\mathbf{M}$, $\mathbb{F}_{p^2}$-squaring $= 2\mathbf{M}$, inversion $= 57\mathbf{M}$, square root $= 2150\mathbf{M}$.

Cubical Ladder (this work)				Miller Loop [CLZ24]				Gain
M	**S**	**A**	**Total**	**M**	**S**	**A**	**Total**	
44 924	21 533	63 373	**66457**	44 542	27 209	63 298	**71751**	7.34%

whether $[p+1]P = 0_E$ for a random $P \in E(\mathbb{F}_p)$ takes only 4853 $\mathbb{F}_p$-operations, with a failure probability of only $O(p^{-1/2})$.

7 Software implementations

In addition to the SageMath implementation for the precise operation counts used in the tables above, we have also implemented versatile proof-of-concept software packages in both SageMath and in Rust. Both implementations are available from the following GitHub repository:

`https://github.com/GiacomoPope/cubical-pairings`.

SageMath. The SageMath implementation has been designed for the general case of $E/\mathbb{F}_q$ and is aligned to the algorithms we present in this paper. We have included detailed comments throughout the code and edge cases are carefully handled.

Rust. The Rust implementation is inspired by the application of cubical pairings in isogeny-based cryptography. It has been written to be efficient and constant-time, including several of the optimisations presented in Section 5.4. In this code, the base field is assumed to be $\mathbb{F}_{p^2}$ (as it is in many isogeny-based applications).

Platform and measurement setup. All running times were captured on an Intel Core i7-9750H CPU with a clock speed of 2.6Ghz, with turbo-boost disabled for stable measurement. The benchmarking itself was handled by the `criterion.rs` crate, which computes an average time by repeating the computation within a time-window determined by evaluation time (with a minimum window of ten seconds). The Rust code was compiled with `rustc 1.87.0-nightly`. All finite field arithmetic was generated without assembly optimisations, but the use of the compiler flag `-C target-cpu=native` allows for CPU-specific intrinsics. In particular, for our benchmarking machine, this allows the use of the `mulx` op-code for efficient 64-bit word multiplication with carries.

Performance results. To pair with the operation counts given in Section 6, we include concrete benchmarks for

- degree-2^n pairings for the SQIsign parameter sets in Table 5,
- change-of-basis computations as described in Section 6.1 in Table 6, and
- degree-2^n and degree-3^n pairings for the SIKE parameter sets in Table 7.

As expected, the cost of a Tate pairing is approximately half that of a Weil pairing in each case. For the degree-2^{e_a} pairings, we use an optimised ladder skipping one call to cADD per bit of the degree, and this is seen in the faster timings compared to the degree 3^{e_b} of similar bit length for the SIKE primes.

For the change-of-basis algorithm, we need to compute five Tate pairings, one of order 2^f and four of order 2^e, as well as some pre-computations and then the final four discrete logs of order 2^e. The total cost of the pairings can be reduced by computing $[2^e]R, [2^e]R+P$ and $[2^e]R-Q$ in a single loop, saving $2 \cdot (e-1)$ cDBL calls. This is seen in the weighting between the costs in Tables 5 and 6.

Table 5: Running times for degree-2^n pairings with the SQIsign parameter sets targeting NIST security levels I, III and V. Times were recorded on an Intel Core i7-9750H CPU with a clock speed of 2.6Ghz and turbo-boost disabled.

	Characteristic	Degree	Weil Pairing	Tate Pairing
p_1	$5 \cdot 2^{248} - 1$	2^{248}	0.31 ms	0.16 ms
p_3	$65 \cdot 2^{376} - 1$	2^{376}	1.01 ms	0.51 ms
p_5	$27 \cdot 2^{500} - 1$	2^{500}	2.40 ms	1.21 ms

Table 6: Running times for SQIsign change-of-basis for points $\langle P, Q\rangle = E[2^f]$ and $\langle R, S\rangle = E[2^e]$ with $e = f/2$ (the typical values used in the process). Times were recorded on an Intel Core i7-9750H CPU with a clock speed of 2.6Ghz and turbo-boost disabled.

	Characteristic	$E[2^f]$	$E[2^e]$	Change of Basis
p_1	$5 \cdot 2^{248} - 1$	2^{248}	2^{124}	0.76 ms
p_3	$65 \cdot 2^{376} - 1$	2^{376}	2^{188}	2.45 ms
p_5	$27 \cdot 2^{500} - 1$	2^{500}	2^{250}	6.13 ms

Table 7: Running times for degree-2^n and degree-3^n pairings with the SIKE parameter sets targeting NIST security levels I, III and V. Times were recorded on an Intel Core i7-9750H CPU with a clock speed of 2.6Ghz and turbo-boost disabled.

	Characteristic	Degree	Weil Pairing	Tate Pairing
p_{434}	$2^{216} \cdot 3^{137} - 1$	2^{216}	1.01 ms	0.61 ms
		3^{137}	1.55 ms	0.87 ms
p_{610}	$2^{305} \cdot 3^{192} - 1$	2^{305}	2.84 ms	1.67 ms
		3^{192}	4.40 ms	2.47 ms
p_{751}	$2^{372} \cdot 3^{239} - 1$	2^{372}	5.11 ms	3.05 ms
		3^{239}	7.84 ms	4.40 ms

References

[AAA+25] Marius A. Aardal et al. SQIsign 2.0: Algorithm specifications and supporting documentation. Technical report, 2025.

[AHG23] Diego F. Aranha, Youssef El Housni, and Aurore Guillevic. A survey of elliptic curves for proof systems. *Des. Codes Cryptogr.*, 91(11):3333–3378, 2023. DOI: 10.1007/S10623-022-01135-Y. URL: https://doi.org/10.1007/s10623-022-01135-y.

[BDD+24] Andrea Basso, Pierrick Dartois, Luca De Feo, Antonin Leroux, Luciano Maino, Giacomo Pope, Damien Robert, and Benjamin Wesolowski. SQIsign2D-west - the fast, the small, and the safer. In pages 339–370, 2024. DOI: 10.1007/978-981-96-0891-1_11.

[BF01] Dan Boneh and Matthew K. Franklin. Identity-based encryption from the Weil pairing. In pages 213–229, 2001. DOI: 10.1007/3-540-44647-8_13.

[BGS22] Gustavo Banegas, Valerie Gilchrist, and Benjamin Smith. Efficient supersingularity testing over $\mathbb{F}_p$ and CSIDH key validation. *Mathematical Cryptology*, 2(1):21–35, 2022.

[BLS04] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the Weil pairing. 17(4):297–319, September 2004. DOI: 10.1007/s00145-004-0314-9.

[Bre83] Lawrence Breen. *Fonctions thêta et théoreme du cube*, volume 980. Springer, 1983.

[CCC+24] Fabio Campos, Jorge Chávez-Saab, Jesús-Javier Chi-Domínguez, Michael Meyer, Krijn Reijnders, Francisco Rodríguez-Henríquez, Peter Schwabe, and Thom Wiggers. Optimizations and practicality of high-security CSIDH. 1(1):5, 2024. DOI: 10.62056/anjbksdja.

[CD23] Wouter Castryck and Thomas Decru. An efficient key recovery attack on SIDH. In pages 423–447, 2023. DOI: 10.1007/978-3-031-30589-4_15.

[CHM+23] Wouter Castryck, Marc Houben, Simon-Philipp Merz, Marzio Mula, Sam van Buuren, and Frederik Vercauteren. Weak instances of class group action based cryptography via self-pairings. In pages 762–792, 2023. DOI: 10.1007/978-3-031-38548-3_25.

[CHMR25] Fabio Campos, Andreas Hellenbrand, Michael Meyer, and Krijn Reijnders. dCTIDH: fast and deterministic CTIDH. Cryptology ePrint Archive, Paper 2025/107, 2025. URL: https://eprint.iacr.org/2025/107.

[CJL+17] Craig Costello, David Jao, Patrick Longa, Michael Naehrig, Joost Renes, and David Urbanik. Efficient compression of SIDH public keys. In pages 679–706, 2017. DOI: 10.1007/978-3-319-56620-7_24.

[CLM+18] Wouter Castryck, Tanja Lange, Chloe Martindale, Lorenz Panny, and Joost Renes. CSIDH: an efficient post-quantum commutative group action. In pages 395–427, 2018. DOI: 10.1007/978-3-030-03332-3_15.

[CLZ24] Shiping Cai, Kaizhan Lin, and Chang-An Zhao. Pairing optimizations for isogeny-based cryptosystems. Cryptology ePrint Archive, Report 2024/575, 2024. URL: https://eprint.iacr.org/2024/575.

[CR24] Maria Corte-Real Santos and Krijn Reijnders. Return of the Kummer: a toolbox for genus-2 cryptography. Cryptology ePrint Archive, Paper 2024/948, 2024. URL: https://eprint.iacr.org/2024/948.

[CS18] Craig Costello and Benjamin Smith. Montgomery curves and their arithmetic - the case of large characteristic fields. 8(3):227–240, September 2018. DOI: 10.1007/s13389-017-0157-6.

[DJP14] Luca De Feo, David Jao, and Jérôme Plût. Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies. *Journal of Mathematical Cryptology*, 8(3):209–247, 2014.

[DKL+20] Luca De Feo, David Kohel, Antonin Leroux, Christophe Petit, and Benjamin Wesolowski. SQISign: compact post-quantum signatures from quaternions and isogenies. In pages 64–93, 2020. DOI: 10.1007/978-3-030-64837-4_3.

[Dol18] Javad Doliskani. On division polynomial PIT and supersingularity. *Applicable Algebra in Engineering, Communication and Computing*, 29(5):393–407, 2018.

[FR94] Gerhard Frey and Hans-Georg Rück. A remark concerning m-divisibility and the discrete logarithm in the divisor class group of curves. *Mathematics of computation*, 62(206):865–874, 1994.

[Gro72] Alexandre Grothendieck. *Groupes de Monodromie en Géométrie Algébrique: SGA 7*. Springer-Verlag, 1972.

[HSV06] Florian Hess, Nigel P. Smart, and Frederik Vercauteren. The eta pairing revisited. *IEEE Trans. Inf. Theory*, 52(10):4595–4602, 2006. DOI: 10.1109/TIT.2006.881709. URL: https://doi.org/10.1109/TIT.2006.881709.

[JAC+22] David Jao, Reza Azarderakhsh, Matthew Campagna, Craig Costello, Luca De Feo, Basil Hess, Amir Jalali, Brian Koziel, Brian LaMacchia, Patrick Longa, Michael Naehrig, Joost Renes, Vladimir Soukharev, David Urbanik, Geovandro Pereira, Koray Karabina, and Aaron Hutchinson. SIKE. Technical report, National Institute of Standards and Technology, 2022. available at https://csrc.nist.gov/Projects/post-quantum-cryptography/round-4-submissions.

[Jou00] Antoine Joux. A one round protocol for tripartite diffie-hellman. In Wieb 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. DOI: 10.1007/10722028_23. URL: https://doi.org/10.1007/10722028%5C_23.

[Lic69] Stephen Lichtenbaum. Duality theorems for curves over p-adic fields. *Inventiones mathematicae*, 7(2):120–136, 1969.

[LR16] David Lubicz and Damien Robert. Arithmetic on abelian and Kummer varieties. *Finite Fields and Their Applications*, 39:130–158, May 2016. DOI: 10.1016/j.ffa.2016.01.009. eprint: 2014/493, HAL: hal-01057467.

[Mil04] Victor S Miller. The Weil pairing, and its efficient calculation. *Journal of cryptology*, 17(4):235–261, 2004.

[MMP+23] Luciano Maino, Chloe Martindale, Lorenz Panny, Giacomo Pope, and Benjamin Wesolowski. A direct key recovery attack on SIDH. In pages 448–471, 2023. DOI: 10.1007/978-3-031-30589-4_16.

[Mon87] Peter L Montgomery. Speeding the Pollard and elliptic curve methods of factorization. *Mathematics of computation*, 48(177):243–264, 1987.

[Mor85] L. Moret-Bailly. *Pinceaux de variétés abéliennes*. Société mathématique de France, 1985.

[MS24] Joseph Macula and Katherine E. Stange. Extending class group action attacks via sesquilinear pairings. Cryptology ePrint Archive, Paper 2024/880, 2024. URL: https://eprint.iacr.org/2024/880.

[MVO91] Alfred Menezes, Scott A. Vanstone, and Tatsuaki Okamoto. Reducing elliptic curve logarithms to logarithms in a finite field. In pages 80–89, 1991. DOI: 10.1145/103418.103434.

[NR19] Michael Naehrig and Joost Renes. Dual isogenies and their application to public-key compression for isogeny-based cryptography. In pages 243–272, 2019. DOI: 10.1007/978-3-030-34621-8_9.

[Rei23] Krijn Reijnders. Effective pairings in isogeny-based cryptography. In pages 109–128, 2023. DOI: 10.1007/978-3-031-44469-2_6.

[Rob23] Damien Robert. Breaking SIDH in polynomial time. In pages 472–503, 2023. DOI: 10.1007/978-3-031-30589-4_17.

[Rob24] Damien Robert. Fast pairings via biextensions and cubical arithmetic. April 2024. eprint: 2024/517, HAL: hal-04848028.

[Sfe24] Alessandro Sferlazza. Hyperelliptic biextension pairings. 2024. URL: https://github.com/sferl/theta-pairings-dim2. SageMath 10.3 library.

[Sta03] Martijn Stam. *Speeding up subgroup cryptosystems*. PhD thesis, Technische Universiteit Eindhoven, 2003.

[Sta07] Katherine E Stange. The Tate pairing via elliptic nets. In *Pairing-Based Cryptography–Pairing 2007: First International Conference, Tokyo, Japan, July 2-4, 2007. Proceedings 1*, pages 329–348. Springer, 2007.

[Sta08] Katherine Stange. *Elliptic nets and elliptic curves*. PhD thesis, Brown University, 2008. URL: https://repository.library.brown.edu/studio/item/bdr:309/PDF/.

[Tat62] John Tate. Duality theorems in Galois cohomology over number fields. In *Proc. Internat. Congr. Mathematicians (Stockholm, 1962)*, pages 288–295, 1962.

[Wei40] André Weil. Sur les fonctions algébriquesa corps de constantes fini. *CR Acad. Sci. Paris*, 210(1940):592–594, 1940.

A Cubical arithmetic in a nutshell

The mathematical results used in the main body of this paper are proven, in much more detail, in [Rob24]. This appendix—which contains no new results—provides a short introduction to the theory, and convenient reference for some of the main results and proofs.

We start with level-1 cubical points in Appendix A.1, discussing their arithmetic properties in Appendices A.2 and A.3. We then describe how cubical arithmetic allows us to compute functions with prescribed divisors in Appendix A.5, which allows us to compute pairings with cubical arithmetic in Appendix A.6. Finally, we move to level-2 cubical points in Appendix A.7, from which we (re)derive the results used in the main text.

So far, we have treated cubical arithmetic as a way to keep track of the extra "hidden" information in affine representatives of projective coordinates. To formalize this, we need to distinguish between the "standard" elliptic curve points, where we only care about their projective coordinates up to some factor $\lambda \in \mathbb{F}_q^*$, and enhanced "cubical points", where we care about the exact values of these coordinates.

This projective factor λ that we track for each point $P \in E$ is be determined by some (projective) coordinate Z. By carefully analyzing this coordinate, we may compute pairings (and even more) using cubical arithmetic. The choice of Z determines the type of cubical arithmetic we use. In this paper, we use Z_n, a nontrivial section of the line bundle associated to the divisor $n(0_E)$ for some positive integer n. Cubical points determined by Z_n are called cubical points of level-n. In practice, we take $Z_n = Z_1^n$, so if we have two level-1 cubical points that differ by a projective factor λ, then their associated level-n cubical points differ by the projective factor λ^n.

A.1 Cubical points of level 1

Let $E/\mathbb{F}_q : y^2 = x^3 + a_2x^2 + a_4x + a_6$ be an elliptic curve, and let Z_1 be the projective coordinate associated to a non-trivial section of the divisor (0_E), meaning that Z_1 has a zero of order 1 at 0_E. We say that Z_1 is a *coordinate* of level-1.[4]

Definition 1. Let $P = (x_P, y_P)$ be a nonzero point on an elliptic curve $E/\mathbb{F}_q$. A *level-1 cubical point* $\widetilde{P}$ above P is the choice of a value $Z_1(\widetilde{P}) \in \mathbb{F}_q^*$, so $\widetilde{P}$ corresponds to the pair $(P, Z_1(\widetilde{P}))$.

[4]We warn the reader that Z_1 is not a function on E. Instead, to get a function on E, we can use the affine coordinate $1 = Z_1/Z_1$ (which is not interesting).

Remark 6. We must modify Definition 1 when $P = 0_E$, because $Z_1(0_E) = 0$ by construction (as Z_1 has a zero of order 1 at 0_E). To define cubical points above 0_E, we divide by a fixed uniformizer at 0_E, say x/y, to get a scalar factor in $\mathbb{F}_q^*$. More precisely: a cubical point is a pair $(0_E, (Z_1/(x/y))(\widetilde{0}_E))$, where $(Z_1/(x/y))(0_E)$ lies in $\mathbb{F}_q^*$. Throughout this paper, we normalize $\widetilde{0}_E$ to satisfy $(Z_1/(x/y))(\widetilde{0}_E) = 1$.[5]

A.2 Cubical arithmetic

The group law $R = P + Q$ on the elliptic curve E lifts to an arithmetic law on cubical points $\widetilde{P}$, $\widetilde{Q}$ and $\widetilde{R}$ called *cubical arithmetic*. This arithmetic law is *not* a group law, but we will see that it shares (and generalizes) features of x-only Kummer arithmetic.

First, we define the cubical inversion by $Z_1(-\widetilde{P}) = -Z_1(\widetilde{P})$.

For points $P_1, P_2 \in E$, let g_{P_1,P_2} denote any function with divisor $(-P_1 - P_2) + (0_E) - (-P_1) - (-P_2)$. Given $P_3 \in E$, we define

$$\mathrm{cub}_1(P_1, P_2, P_3) := g_{P_1,P_2}((P_3) - (0_E)) = \frac{g_{P_1,P_2}(P_3)}{g_{P_1,P_2}(0_E)}\,. \tag{3}$$

This quantity is independent of the choice of g_{P_1,P_2}; it can be computed explicitly as

$$\mathrm{cub}_1(P_1, P_2, P_3) = \frac{l_{P_1,P_2}(P_3)}{(x(P_3) - x(P_1))(x(P_3) - x(P_2))} = \frac{x(P_1 + P_2) - x(P_3)}{l_{P_1,P_2}(-P_3)}.$$

We can now define the cubical arithmetic law.

Definition 2. Given points P_1, P_2, P_3, we may form a cube 0_E, P_1, P_2, P_3, $P_2 + P_3$, $P_1 + P_3$, $P_1 + P_2$, $P_1 + P_2 + P_3$, as in Figure 1. We say the associated level-1 cubical points $\widetilde{0_E}$, $\widetilde{P_1}$, $\widetilde{P_2}$, $\widetilde{P_3}$, $\widetilde{P_2 + P_3}$, $\widetilde{P_1 + P_3}$, $\widetilde{P_1 + P_2}$, $\widetilde{P_1 + P_2 + P_3}$ form a *cubical cube* if

$$\mathrm{cub}_1(P_1, P_2, P_3) = \frac{Z_1(P_1 \widetilde{+ P_2 + P_3}) \cdot Z_1(\widetilde{P_1}) \cdot Z_1(\widetilde{P_2}) \cdot Z_1(\widetilde{P_3})}{Z_1(\widetilde{0_E}) \cdot Z_1(\widetilde{P_2 + P_3}) \cdot Z_1(\widetilde{P_1 + P_3}) \cdot Z_1(\widetilde{P_1 + P_2})}\,. \tag{4}$$

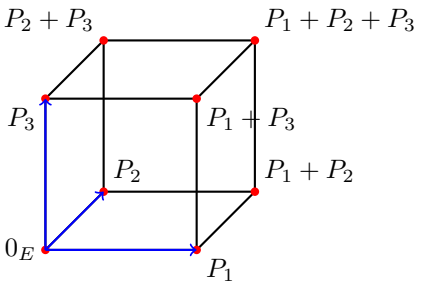

Figure 1: A cube of points P_1, P_2, P_3.

Remark 7. Technically, Eq. (4) is not well-defined because $Z_1(\widetilde{0_E}) = 0$ and we evaluate g_{P_1,P_2} at a zero 0_E; but it makes sense if we multiply both sides by a uniformizer, like x/y. As mentioned above, we always choose $\widetilde{0}_E$ such that $(Z_1/(x/y))(\widetilde{0}_E) = 1$.

[5] Note that Z_1 is a cubical function of level-1 with a zero or order 1 at 0_E, and x/y is a standard function with a zero of order 1 at 0_E, so their quotient is a cubical function with no poles or zeroes at 0_E. If we think in terms of rigidifications of line bundles, then the choice of $\widetilde{0}_E$, i.e. of rigidification, allows us to interpret Z as a standard function ϕ_Z on E locally on 0_E, i.e. as an element of $\mathcal{O}_{E,0_E}$, and $\phi_Z/(x/y)$ is a well-defined element of $\mathcal{O}^*_{E,0_E}$; and the normalization condition is that its value at 0_E should be 1.

The crucial property—whence the name *cubical arithmetic*—is that Eq. (4) allows us to derive $Z_1(P_1 + \widetilde{P_2 + P_3})$ from the seven other vertices of the cube. Similarly, given seven vertices of *some* cube, we can compute the eighth cubical vertex: that is, the vertex R with the correct value of $Z_1(\widetilde{R}) \in \mathbb{F}_q^*$.

The most important cube we use in this article is given by $(P_1, P_2, P_3) = (P, Q, -Q)$, which results in a cube with vertices 0_E, P, Q, $-Q$, 0_E, $P - Q$, $P + Q$, and P. Using Eq. (4), we can derive $Z_1(\widetilde{P+Q}) \cdot Z_1(\widetilde{P-Q})$.

Lemma 1. *Given $Z_1(\widetilde{P})$, $Z_1(\widetilde{Q})$, and $(x(Q) - x(P))$, the cube above defines* cubical differential addition*:*

$$Z_1(\widetilde{P+Q}) \cdot Z_1(\widetilde{P-Q}) = Z_1(\widetilde{P})^2 \cdot Z_1(\widetilde{Q})^2 \cdot (x(Q) - x(P)). \tag{5}$$

Specializing further to the case $P = Q$, we obtain cubical doubling*:*

$$Z_1(2\widetilde{P}) = Z_1(\widetilde{P})^4 \cdot 2y(P). \tag{6}$$

Proof. For the cubical differential addition, by Definition 2, we have

$$\frac{Z_1(\widetilde{P+Q})Z_1(\widetilde{P-Q})Z_1(\widetilde{0}_E)Z_1(\widetilde{0}_E)}{Z_1(\widetilde{P})Z_1(\widetilde{P})Z_1(\widetilde{Q})Z_1(\widetilde{-Q})} = \frac{g_{Q,-Q}(0_E)}{g_{Q,-Q}(P)}.$$

The result follows from the fact that $Z_1(-\widetilde{Q}) = -Z_1(\widetilde{Q})$, and that if we take $g_{Q,-Q} = 1/(x - x(Q))$, then it is normalized at infinity—that is, $(g_{Q,-Q}/(x/y)^2)(0_E) = 1$. Our normalization condition on $\widetilde{0}_E$ gives $(Z_1^2/g_{Q,-Q})(\widetilde{0}_E) = 1$.

For cubical doubling we have an extra 0_E in the numerator, so we need to compute the inverse of $Z(\widetilde{0}_E) \cdot g_{P,-P}(P)$. As above, we take $g_{P,-P} = 1/(x - x(P))$, which is normalized at 0_E. To compute the evaluation, we let $g' = t_P^* g_{P,-P}$, i.e., $g'(R) = g_{P,-P}(R + P)$, so that $Z(\widetilde{0}_E) \cdot g_{P,-P}(P) = (g' \cdot x/y)(0_E)$. The formulæ for the addition law give

$$\frac{1}{g' \cdot x/y} = \left(\frac{(y - y(P))^2}{(x - x(P))^2} - x - 2x(P)\right)/(x/y),$$

so $((g' \cdot x/y)(0_E))^{-1} = -2y(P)$. The result follows because $Z_1(-\widetilde{P}) = -Z_1(\widetilde{P})$. □

A.3 Properties of cubical arithmetic

Theorem 1 summarizes the most useful properties of cubical arithmetic for this work.

Theorem 1. *Let $P_1, P_2, P_3, P_4 \in E$.*

1. *Neutrality:* $\mathrm{cub}_1(0_E, 0_E, 0_E) = 1$.
2. *Commutativity:* $\mathrm{cub}_1(\sigma(P_1, P_2, P_3)) = \mathrm{cub}_1(P_1, P_2, P_3)$ *for all* $\sigma \in \mathfrak{S}_3$.
3. *Associativity:*

$$\mathrm{cub}_1(P_1 + P_2, P_3, P_4) \cdot \mathrm{cub}_1(P_1, P_2, P_4) = \mathrm{cub}_1(P_1, P_2 + P_3, P_4) \cdot \mathrm{cub}_1(P_2, P_3, P_4).$$

4. *Anti-symmetry:* $\mathrm{cub}_1(P_1, P_2, -P_1 - P_2) = -1$.

Proof. Let g_{P_1,P_2} be a function with divisor $(-P_1 - P_2) + (0_E) - (-P_1) - (-P_2)$, which this time we assume is normalized at 0_E by the condition $g_{P_1,P_2}/(x/y)(0_E) = 1$. Eq. (3) and Definition 2 give $\mathrm{cub}_1(P_1, P_2, P_3) = g_{P_1,P_2}(P_3)$, and neutrality follows for $P_1 = P_2 = P_3 = 0_E$. Similarly, we can rewrite the three other conditions as

- Commutativity: $g_{P_1,P_2}(P_3) = g_{P_2,P_3}(P_1) = g_{P_3,P_1}(P_2)$
- Associativity: $g_{P_1+P_2,P_3} \cdot g_{P_1,P_2} = g_{P_1,P_2+P_3} \cdot g_{P_2,P_3}$
- Anti-symmetry: $g_{P_1,P_2}(-P_1 - P_2) = -1$.[6]

Associativity follows quickly from the fact that the LHS and RHS have the same divisor and are both normalized. Commutativity and anti-symmetry are more delicate: both follow from much more general results involving symmetric biextensions [Gro72] and Σ-cubical torsor structures on abelian varieties [Bre83], though in our case they could also be proven using symbolic algebra software. [Rob24] gives a full exposition. □

Corollary 1 (Cubical $\mathbb{Z}$-linear combinations)**.** *Let $P_1, \dots, P_m \in E$, and choose cubical points $\widetilde{P}_i, \widetilde{P_i + P_j}$ for all $1 \le i, j \le m$. Then, for $n_1, \dots, n_m \in \mathbb{Z}$, we are free to compute a cubical point above the elliptic point $\sum n_i P_i$ using any choice of cubes and inversions: we always obtain the same cubical point $\widetilde{\sum n_i P_i}$.*

Since the cubical point $\widetilde{\sum n_i P_i}$ of Corollary 1 is independent of the n_i, we denote it by $\sum n_i \widetilde{P}_i$.

Proof of the corollary. When $n_i \ge 0$ and we don't use inversions, this follows from commutativity and associativity (Theorem 1). For the general case, when $n_i \in \mathbb{Z}$, we also need inversions to compute $\sum n_i \widetilde{P}_i$. Then, the anti-symmetry condition of Theorem 1 shows that the result is independent of the choice of point to invert. Indeed, for the cube with $P_3 = -P_1 - P_2$, Eq. (4) and anti-symmetry yield

$$\frac{Z_1(\widetilde{P_1})Z_1(\widetilde{P_2})Z_1(\widetilde{-P_1 - P_2})}{Z_1(\widetilde{-P_1})Z_1(\widetilde{-P_2})Z_1(\widetilde{P_1 + P_2})} = \mathrm{cub}_1(P_1, P_2, -P_1 - P_2) = -1\,;$$

and the inversion formula $Z_1(-\widetilde{P}) = -Z_1(\widetilde{P})$ gives the same result. □

We warn the reader that $\sum n_i \widetilde{P}_i$ depends not only on the choices of $\widetilde{P}_i$, but also the $\widetilde{P_i + P_j}$. Different choices scale the resulting cubical point by a projective factor $\lambda \in \mathbb{F}_q^*$ given by [Sta08, Theorem 10.1.1], rephrased in cubical terms in [Rob24, Lemma 4.7]. Lemma 2 gives λ in the notation of this appendix.

Lemma 2. *Let $\widetilde{P}_i{}', \widetilde{P_i + P_j}{}'$ be other choices of cubical points above $P_i, P_i + P_j$. If λ_i, $\lambda_{i,j} \in \mathbb{F}_q^*$ are such that $Z_1(\widetilde{P}_i{}') = \lambda_i \cdot Z_1(\widetilde{P}_i)$ and $Z_1(\widetilde{P_i + P_j}{}') = \lambda_i \lambda_j \lambda_{i,j} \cdot Z_1(\widetilde{P_i + P_j})$, then*

$$Z_1\left(\sum n_i \widetilde{P}_i{}'\right) = \lambda \cdot Z_1\left(\sum n_i \widetilde{P}_i\right) \quad \text{where} \quad \lambda := \prod_{i=1}^{m} \lambda_i^{n_i^2} \prod_{1 \le i < j \le m} \lambda_{i,j}^{n_i n_j}.$$

A.4 Translated cubes

For completeness, we also mention that we can extend the cubical law from cubes to translated cubes: P_0, $P_0 + P_1$, $P_0 + P_2$, $P_0 + P_3$, $P_0 + P_2 + P_3$, $P_0 + P_1 + P_3$, $P_0 + P_1 + P_2$, $P_0 + P_1 + P_2 + P_3$ as in Figure 2, by the formula:

$$\frac{\mathrm{cub}_1(P_1, P_2, P_0 + P_3)}{\mathrm{cub}_1(P_1, P_2, P_0)} = \frac{Z_1(P_0 + \widetilde{P_1 + P_2 + P_3}) \cdot Z_1(\widetilde{P_0 + P_1}) \cdot Z_1(\widetilde{P_0 + P_2}) \cdot Z_1(\widetilde{P_0 + P_3})}{Z_1(\widetilde{P_0}) \cdot Z_1(P_0 + \widetilde{P_2 + P_3}) \cdot Z_1(P_0 + \widetilde{P_1 + P_3}) \cdot Z_1(P_0 + \widetilde{P_1 + P_2})}.$$

[6] Or, more rigorously, that if $g' = t^*_{-P_1-P_2}(g_{P_1,P_2})$, then $(g'/g_{P_1,P_2})(0_E) = -1$.

Indeed, the formula for the translated cube can be found by looking at the two cubes generated by P_1, P_2, P_3 and $P_1, P_2, P_0 + P_3$ respectively. We also have that $\frac{\mathrm{cub}_1(P_1,P_2,P_0+P_3)}{\mathrm{cub}_1(P_1,P_2,P_0)} = g_{P_1,P_2}((P_0 + P_3) - (P_0))$.

We remark that the translated cube can be given a more symmetric form in its inputs by setting $U_1 = P_0 + P_1, U_2 = P_0 + P_2, U_3 = P_0 + P_3, U_4 = P_0 + P_1 + P_2 + P_3$, $W = 2P_0 + P_1 + P_2 + P_3$ (so that $U_1 + U_2 + U_3 + U_4 = 2W$), and $V_1 = W - U_2 = P_0 + P_2 + P_3, V_2 = W - U_3 = P_0 + P_1 + P_3, V_3 = W - U_4 = P_0 + P_1 + P_2, V_4 = W - U_1 = P_0$. Conversely, from $U_1, U_2, U_3, U_4, V_1, V_2, V_3, V_4$ we recover $P_0 = V_4$, $P_1 = U_1 - V_4, P_2 = U_2 - V_4, P_3 = U_3 - V_4$.

The cube arithmetic then takes the form

$$\frac{\mathrm{cub}_1(U_1 - V_4, U_2 - V_4, U_3)}{\mathrm{cub}_1(U_1 - V_4, U_2 - V_4, V_4)} = \frac{Z_1(\widetilde{U_1}) \cdot Z_1(\widetilde{U_2}) \cdot Z_1(\widetilde{U_3}) \cdot Z_1(\widetilde{U_4})}{Z_1(\widetilde{V_1}) \cdot Z_1(\widetilde{V_2}) \cdot Z_1(\widetilde{V_3}) \cdot Z_1(\widetilde{V_4})}.$$

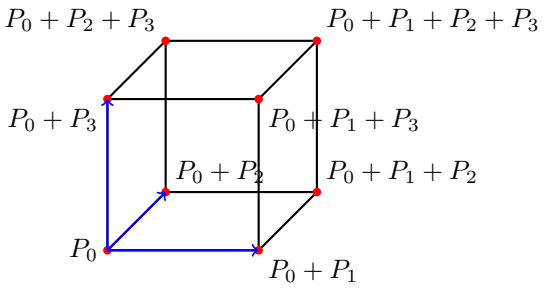

Figure 2: A cube of points P_1, P_2, P_3 translated by P_0

A.5 Elliptic functions

Cubical arithmetic gives us an alternative to Miller's algorithm for constructing functions on elliptic curves with prescribed divisors.

Lemma 3. *Let $P_1, \ldots, P_m \in E$, with associated level-1 cubical points $\widetilde{P}_i$ and $\widetilde{P_i + P_j}$. Suppose we have m linear combinations $\sum_{i=1}^{m} u_{j,i} P_i$ for $u_{j,i} \in \mathbb{Z}$. If there exist $v_j \in \mathbb{Z}$ and $u_{j,0} \in \mathbb{Z}$ such that $\sum_{j=1}^{r} v_j u_{j,0} u_{j,i} = 0$ for $0 \le i \le m$, then for any $R \in E$, the value*

$$\prod_{j=1}^{r} Z_1(u_{j,0}\widetilde{R} + \sum_{i=1}^{m} u_{j,i}\widetilde{P}_i)^{v_j}$$

is independent of the choice of $\widetilde{R}$ and $\widetilde{R + P_i}$.

Proof. For ease of notation, we consider R as P_0 with respect to Lemma 2, and let $\widetilde{P_0}'$ and $\widetilde{P_0 + P_i}'$ be other choices of cubical points above P_0 and $P_0 + P_i$, with $\widetilde{P_0}' = \lambda_0 \widetilde{P_0}$ and $\widetilde{P_0 + P_i}' = \lambda_0 \lambda_{0,i} \cdot \widetilde{P_0 + P_i}$ for some scalars λ in $\mathbb{F}_q^*$. We want to show that the condition $\sum_{j=1}^{r} v_j u_{j,0} u_{j,i} = 0$ for all i implies that the differences in λ_0 and $\lambda_{0,i}$ vanish for the overall product. By Lemma 2, we have

$$\frac{Z_1(u_{j,0}\widetilde{P_0}' + \sum_{i=1}^{m} u_{j,i}\widetilde{P_i}')}{Z_1(u_{j,0}\widetilde{P_0} + \sum_{i=1}^{m} u_{j,i}\widetilde{P_i})} = \lambda_0^{u_{j,0}^2} \prod_{i=1}^{m} \lambda_{0,i}^{u_{j,0}u_{j,i}},$$

and therefore

$$\prod_{j=1}^{r}\left(\frac{Z_1(u_{j,0}\widetilde{P_0}' + \sum_{i=1}^{m} u_{j,i}\widetilde{P_i}')}{Z_1(u_{j,0}\widetilde{P_0} + \sum_{i=1}^{m} u_{j,i}\widetilde{P_i})}\right)^{v_j} = \prod_{j=1}^{r}\left(\lambda_0^{u_{j,0}^2}\prod_{i=1}^{m}\lambda_{0,i}^{u_{j,0}u_{j,i}}\right)^{v_j}$$
$$= \left(\lambda_0^{\sum_{j=1}^{r} v_j u_{j,0}^2}\prod_{i=1}^{m}\lambda_{0,i}^{\sum_{j=1}^{r} v_j u_{j,0}u_{j,i}}\right) = 1\,,$$

where the last equality uses the condition that each $\sum_{j=1}^{r} v_j u_{j,0}u_{j,i} = 0$. Hence, the product $\prod_{j=1}^{r} Z_1(u_{j,0}\widetilde{P_0} + \sum_{i=1}^{m} u_{j,i}\widetilde{P_i})$ is independent of the choice of $\widetilde{P_0}$ and $\widetilde{P_0 + P_i}$. □

We can now construct a function $f : E \to \mathbb{P}^1$ with a prescribed divisor by mapping $R \in E$ to the projective value defined in Lemma 3, which gives a well-defined map.

Theorem 2. *With the situation as in Lemma 3, that is, we have r elements $u_1 = (u_{1,0}, \ldots, u_{1,m})$, $\ldots$, $u_r = (u_{r,0}, \ldots, u_{r,m})$ in $\mathbb{Z}^{m+1}$, and $v_1, \ldots, v_r \in \mathbb{Z}$ such that for all $i \in \{0, \ldots, m\}$, we have $\sum_{j=1}^{r} v_j u_{j,0}u_{j,i} = 0$. Then the function*

$$f : E \to \mathbb{P}^1, \qquad R \mapsto \prod_{j=1}^{r} Z_1(u_{j,0}\widetilde{R} + \sum_{i=1}^{m} u_{j,i}\widetilde{P_i})^{v_j},$$

is a well-defined function on E with divisor

$$\operatorname{div}(f) = \sum_{j=1}^{r} v_j [u_{j,0}]^*(-\sum_{i=1}^{m} u_{j,i}P_i).$$

Proof. For a complete proof, see [Rob24, § 4.6]. We sketch the main idea here. Lemmas 2 and 3 imply that $f(R)$ does not depend on the choice of $\widetilde{R}$ and $\widetilde{R + P_i}$, so f is well-defined on E. Evaluating f at the linear combinations of P_i gives the divisor in the statement. □

We give two useful examples of Theorem 2, which we will revisit in Appendix A.6.

Example 1. The function

$$f_{P_1,P_2} : R \longmapsto \frac{Z_1(R + \widetilde{P_1 + P_2})Z_1(\widetilde{R})}{Z_1(\widetilde{R + P_1})Z_1(\widetilde{R + P_2})}$$

depends only on the choice of $\widetilde{P_1}, \widetilde{P_2}, \widetilde{P_1 + P_2}$. Its divisor is

$$\operatorname{div} f_{P_1,P_2} = (-P_1 - P_2) + (0_E) - (-P_1) - (-P_2)\,.$$

Example 2. The function

$$f_{\ell,P} : R \longmapsto \frac{Z_1(\ell\widetilde{P} + \widetilde{R})Z_1(\widetilde{R})^{\ell-1}}{Z_1(\widetilde{P + R})^{\ell}}$$

depends only on the choice of $\widetilde{P}$. Its divisor is

$$\operatorname{div} f_{\ell,P} = (-\ell P) + (\ell - 1)(0_E) - \ell(-P)\,.$$

A.6 Pairings from cubical arithmetic

Using Examples 1 and 2, we can reformulate all pairing formulæ in the literature, which are usually expressed in terms of Miller functions, in terms of cubical arithmetic. This is similar to how Stange used elliptic nets to compute pairings in [Sta08, §17; Sta07]—indeed, elliptic nets give an alternative way to compute level-1 cubical arithmetic, and Theorem 2 can be seen as a generalization of [Sta08, § 10.3].

Theorem 3.

- *Let $P, Q \in E[\ell]$, and $\widetilde{P}, \widetilde{Q}, \widetilde{P+Q}$ be arbitrary cubical points above $P, Q, P+Q$. Then*

$$e_{W,\ell}(P,Q) = \frac{Z_1(\ell\widetilde{P}+\widetilde{Q})Z_1(\ell\widetilde{Q})Z_1(\widetilde{P})}{Z_1(\ell\widetilde{P})Z_1(\ell\widetilde{Q}+\widetilde{P})Z_1(\widetilde{Q})}$$

- *Let $P \in E[\ell](\mathbb{F}_q)$ and $Q \in E(\mathbb{F}_q)$, and $\widetilde{P}, \widetilde{Q}, \widetilde{P+Q}$ be arbitrary* rational *cubical points above $P, Q, P+Q$. The non-reduced Tate pairing is*

$$e_{T,\ell}(P,Q) = \frac{Z_1(\ell\widetilde{P}+\widetilde{Q})Z_1(\widetilde{0}_E)}{Z_1(\widetilde{Q})Z_1(\ell\widetilde{P})}$$

Proof. The usual formulæ for the Weil and Tate pairings are $e_{W,\ell}(P,Q) = \frac{f_{\ell,P}((Q)-(0_E))}{f_{\ell,Q}((P)-(0_E))}$ and $e_{T,\ell}(P,Q) = f_{\ell,P}((Q)-(0_E)) \cdot \left(\frac{x}{y}\right)^{\ell}(0_E)$, where the rational uniformizer (x/y) is needed to make the expression well-defined. Example 2 gives expressions for $f_{\ell,P}$ and $f_{\ell,Q}$. (The functions used here and in Section 2.3 have linearly equivalent divisors, thus give the same pairing values.) For instance, we obtain

$$e_{T,\ell}(P,Q) = \frac{Z_1(\ell\widetilde{P}+\widetilde{Q})Z_1(\widetilde{0}_E)}{Z_1(\widetilde{Q})Z_1(\ell\widetilde{P})} \left(\frac{Z_1(\widetilde{P})Z_1(\widetilde{Q})}{Z_1(\widetilde{P+Q})(Z_1/(x/y))(\widetilde{0}_E)}\right)^{\ell}$$

—which gives the formula above, since we assumed the cubical points were defined over $\mathbb{F}_q$, hence $\left(\frac{Z_1(\widetilde{P})Z_1(\widetilde{Q})}{Z_1(\widetilde{P+Q})(Z_1/(x/y))(\widetilde{0}_E)}\right)^{\ell} \in (\mathbb{F}_q^\star)^\ell$. □

Remark 8. Since $[\ell]P = 0_E$ for $P \in E[\ell]$, we must remember that $Z_1(\ell P) = Z_1(0_E) = 0$, and the formulæ above should be understood with respect to some uniformizer. For example, in the Weil pairing, $\frac{Z_1(\ell\widetilde{Q})}{Z_1(\ell\widetilde{P})}$ should be understood as $\frac{Z_1/(x/y)(\ell\widetilde{Q})}{Z_1/(x/y)(\ell\widetilde{P})}$, and in the Tate pairing $\frac{Z_1(\widetilde{0}_E)}{Z_1(\ell\widetilde{P})}$ should be understood as $\frac{Z_1/(x/y)(\widetilde{0}_E)}{Z_1/(x/y)(\ell\widetilde{P})}$.

Similarly, we can obtain translated cubical formulæ from Theorem 3. For the Weil pairing, we use $e_{W,\ell}(P,Q) = \frac{f_{\ell,P}((Q+R)-(R))}{f_{\ell,Q}((P+R)-(R))}$ for any $R \in E$. For the Tate pairing, we use $e_{T,\ell}(P,Q) = f_{\ell,P}((Q+R)-(R))$ for rational points $R \in E(\mathbb{F}_q)$.

A.7 Cubical arithmetic in level 2

Recall that a level-1 cubical point $\widetilde{P}$ is the datum of an elliptic curve point P *and* a "cubical" coordinate $Z_1(\widetilde{P}) \in \mathbb{F}_q^*$. We cannot recover P from the coordinate $Z_1(\widetilde{P})$ alone.

In the main body of this paper we use *level*-2 cubical coordinates, which—unlike level-1 coordinates—*do* allow us to recover the underlying point P up to sign. This lets us compute cubical arithmetic by slightly adapting algorithms for x-only arithmetic on Kummer lines, replacing projective coordinates with cubical coordinates.

Let X_2, Z_2 be the basis of projective coordinates associated to the divisor $2(0_E)$ such that $Z_2 = Z_1^2$ and $x = X_2/Z_2$. These *level*-2 coordinates are our main tool in the sequel, so for ease of notation we write $X = X_2, Z = Z_2$.

Example 3. Let $P = (x_P, y_P) \in E$. A *level-2 cubical point* $\widetilde{P}$ above P is a choice of a value $Z(\widetilde{P}) \in \mathbb{F}_q^*$, where $Z := Z_2 = Z_1^2$. Since $x = X/Z$, choosing $\widetilde{P}$ above P, i.e., choosing $Z(\widetilde{P}) \in \mathbb{F}_q^*$, also determines $X(\widetilde{P})$; so a choice of $\widetilde{P}$ is the same as a choice of affine coordinates $(X(\widetilde{P}), Z(\widetilde{P}))$ above the projective coordinates $(X_P : Z_P)$ of P.

Given a level-2 cubical point $\widetilde{P} = (P, (X(\widetilde{P}), Z(\widetilde{P})))$ above P, the data $(X(\widetilde{P}), Z(\widetilde{P}))$ alone suffices to recover $x(P) = X(\widetilde{P})/Z(\widetilde{P})$, and hence P up to sign. This means that $(X(\widetilde{P}), Z(\widetilde{P}))$ are the coordinates of a "Kummer line cubical point" rather than an elliptic curve cubical point, which is still enough for our applications.

A nice feature is that $\mathrm{cub}_2(P_1, P_2, P_3)$ only depends on $x(P_1)$, $x(P_2)$, $x(P_3)$, $x(P_2+P_3)$, $x(P_1+P_3)$, $x(P_1+P_2)$, and $x(P_1+P_2+P_3)$ by [Rob24, § 4.9.4] so that level-2 cubical arithmetic can be done entirely on the Kummer line. We remark that we can also recover $x(P_1+P_2)$ and $x(P_1+P_2+P_3)$ from $x(P_1)$, $x(P_2)$, $x(P_3)$, $x(P_2+P_3)$, $x(P_1+P_3)$ [LR16].

Another advantage of level-2 cubical coordinates is that since $2(0_E)$ is base-point-free on E, we do not need a special case to define our cubical neutral point, as it is determined by its X-coordinate: $\widetilde{0}_E = (1, 0)$. Indeed, since $Z_1/(x/y)(\widetilde{0}_E) = 1$, we get $X(\widetilde{0}_E) = \frac{Xx^2}{Zy^2}(\widetilde{0}_E)\frac{Zy^2}{x^2}(\widetilde{0}_E) = \frac{x^3}{y^2}(0_E) = 1$.

In the main text we work with level-2 cubical arithmetic, by interpreting (as in Example 3) ordinary projective coordinates $(X_P : Z_P)$ as level-2 cubical points $(X(\widetilde{P}), Z(\widetilde{P}))$ while tracking projective factors. Indeed, if we repeat the derivations of cubical differential addition and doubling in Equation (5) and Equation (6) but for level-2 cubical points, then we recover the formulæ for cADD and cDBL described in Sections 3 and 4.

Remark 9 (The case of twists). Let $E : y^2 = x^3 + a_2x^2 + a_4x + a_6$ be an elliptic curve and $E' : By'^2 = x'^3 + a_2x'^2 + a_4x' + a_6$ be a quadratic twist over a base field k. The two curves are isomorphic over the quadratic extension $k(\alpha)$, where $\alpha^2 = B$, via the map $E \to E',\ (x, y) \mapsto (x', y') = (x, y/\alpha)$.

Cubical arithmetic can be performed on the twist E': the same proof as in Lemma 1, now taking into account that $(y'^2/x'^3)(0_{E'}) = 1/B$, shows that the level 1 cubical function Z_1' attached to E' satisfies

$$Z_1'(\widetilde{P+Q}) \cdot Z_1'(\widetilde{P-Q}) = \frac{1}{B} Z_1'(\widetilde{P})^2 \cdot Z_1'(\widetilde{Q})^2 \cdot (x'(Q) - x'(P)),$$
$$Z_1'(2\widetilde{P}) = \frac{1}{B} Z_1'(\widetilde{P})^4 \cdot 2y'(P).$$

Alternatively, since $(Z_1/(x'/y'))(\widetilde{0}_E) = 1/\alpha$, the normalization condition at $0_{E'}$ for the twist E' is satisfied by the cubical coordinate $Z_1' = \alpha Z_1$. Substituting this Z_1', x', y' in Eqs. (5) and (6) also gives the above formula. We remark how, unlike standard x-only arithmetic, cubical arithmetic does depend on the choice of twist we work on.

The results of Appendix A.10 to compute the Tate pairing via level-2 cubical arithmetic generalize to the case of twists. On E', we must use coordinates $Z_2' = {Z_1'}^2$ and $X_2' = x'Z_2'$, which yield $\widetilde{0}_{E'} = (B, 0)$ and satisfy the following:

$$Z_2'(\widetilde{P+Q}) \cdot Z_2'(\widetilde{P-Q}) = \frac{1}{B^2} Z_2'(\widetilde{P})^2 \cdot Z_2'(\widetilde{Q})^2 \cdot (x'(Q) - x'(P))^2, \tag{7}$$
$$Z_2'(2\widetilde{P}) = \frac{1}{B^2} Z_2'(\widetilde{P})^4 \cdot 4(x'(P)^3 + a_2x'(P)^2 + a_4x'(P) + a_6). \tag{8}$$

Implementors might be tempted to use scaled coordinates $(X_2'', Z_2'') = (X_2'/B, Z_2'/B)$ which get rid of B in Eq. (7) and satisfy $(X_2'', Z_2'')(\widetilde{0}_{E'}) = (1, 0)$. However, the correct computation of the Tate pairing on E' (rather than its square) relies on the fact that Z_2' is the square of a k-rational level-1 cubical coordinate, hence why $Z'' = (Z_1'/\alpha)^2$ should not be used in this context.

A.8 Higher levels

More generally, we can define coordinates of level-n. Fix $Z_n = Z_1^n$, which is a projective coordinate associated to a non-trivial section of the divisor $n(0_E)$.

Definition 3. A *cubical point* $\widetilde{P}$ *of level-n* above an elliptic curve point $P \in E$, with $P = (x_P, y_P) \neq 0_E$, is the choice of a value $Z_n(\widetilde{P}) \in \mathbb{F}_q^*$, where $Z_n \coloneqq Z_1^n$.

Like in level-1, cubical points above 0_E correspond to values of $Z_n/(x/y)^n(\widetilde{0}_E)$, not Z_n. And as in level-1, we always take $\widetilde{0}_E$ such that $Z_n/(x/y)^n(\widetilde{0}_E) = 1$.

Example 4. Let $P = (x_P, y_P) \in E$. Let X_3, Y_3, Z_3 be the basis of projective coordinates associated to the divisor $3(0_E)$ such that $Z_3 = Z_1^3$, $x = X_3/Z_3 = X_2 Z_1$, and $y = Y_3/Z_3$. Then a choice of level-3 cubical point $\widetilde{P}$ above P, i.e., a choice of $Z_3(\widetilde{P}) \in \mathbb{F}_q^*$, is the same as a choice of affine coordinates $(X_3(\widetilde{P}), Y_3(\widetilde{P}), Z_3(\widetilde{P}))$ above the projective coordinates $(X_{3,P} : Y_{3,P} : Z_{3,P})$ of P.

Similar to Definition 2, with $\mathrm{cub}_n(P_1, P_2, P_3) = \mathrm{cub}_1(P_1, P_2, P_3)^n$, we get the level-$n$ cubical arithmetic law:

$$\frac{Z_n(P_1 + \widetilde{P_2 + P_3})Z_n(\widetilde{P_1})Z_n(\widetilde{P_2})Z_n(\widetilde{P_3})}{Z_n(\widetilde{0}_E)Z_n(\widetilde{P_2 + P_3})Z_n(\widetilde{P_1 + P_3})Z_n(\widetilde{P_1 + P_2})} = \mathrm{cub}_n(P_1, P_2, P_3) \tag{9}$$

The anti-symmetry of level 1 becomes symmetry for even levels, and anti-symmetry for odd levels:

$$\mathrm{cub}_n(P_1, P_2, -P_1 - P_2) = \mathrm{cub}_1(P_1, P_2, -P_1 - P_2)^n = (-1)^n\,,$$

and indeed, $Z_n(-\widetilde{P}) = Z_1^n(-\widetilde{P}) = (-Z_1(\widetilde{P}))^n = (-1)^n Z(\widetilde{P})$.

A.9 Cubical translation

An extra tool in level-n cubical arithmetic is the cubical translation by $\widetilde{T}$ for $T \in E[n]$ [Mor85, § I.4; Rob24, § 4.2.6], which acts as a matrix on the level-n cubical coordinates. For instance, if $E : y^2 = x^3 + Ax^2 + x$ is a Montgomery curve, we always have $T = (0 : 1) \in E[2]$. The level-2 cubical translation by $\widetilde{T} = (0, 1)$ is then given by $(X, Z) \mapsto (Z, X)$.

Definition 4. Let $\widetilde{T}$ be a level-n cubical point above a point T of n-torsion. Given a level-n cubical point $\widetilde{P}$, we define $\widetilde{P + T}$ by the formula

$$\frac{Z_n(\widetilde{P + T})Z_n(\widetilde{0}_E)}{Z_n(\widetilde{P})Z_n(\widetilde{T})} = \frac{\mathrm{cub}_{n,T}(0_E)}{\mathrm{cub}_{n,T}(P)}$$

where we recall that $\mathrm{cub}_{n,T}$ is any function with divisor $n(0_E) - n(-T)$.

Example 5. In level-2, we can take $\mathrm{cub}_{2,T} = 1/(x - x(T))$ and, since it is normalized, the formula becomes $Z(\widetilde{P + T}) = Z(\widetilde{P})Z(\widetilde{T})(x(P) - x(T))$. For a Montgomery curve, with $\widetilde{T} = (0, 1)$, we obtain $Z(\widetilde{P + T}) = Z(\widetilde{P})\frac{X}{Z}(P) = X(\widetilde{P})$, and so $X(\widetilde{P + T}) = Z(\widetilde{P + T})x(P + T) = X(\widetilde{P})/x(P) = Z(\widetilde{P})$. We recover the translation formula above.

Note that working in level-n, applying Theorem 2 with $Z_n = Z_1^n$ instead of Z_1, we can only construct n-th power of rational functions on E. If $f : E \to \mathbb{P}^1$ is a rational function with divisor $\sum nm_i(P_i)$, it is an n-th power (possibly over an extension) if and only if $T \coloneqq \sum m_i P_i$ is the neutral point 0_E. In general, since $\sum nm_i P_i = 0_E$, T is only a point of n-torsion. For instance, when $n = 2$, the function $x - x(T)$ of divisor $2(T) - 2(0_E)$ is not a square. Thankfully, we can use the cubical translation to tackle this case.

Theorem 4. *Suppose we are given $P_1, \dots, P_m \in E$, and also level-n cubical points $\widetilde{P}_i$, $\widetilde{P_i + P_j}$. Suppose also that we are given level-n cubical points $\widetilde{T}_i$ where $T_i \in E[n]$.*

Let $u_1 = (u_{1,0}, \dots, u_{1,m})$, $\dots$, $u_r = (u_{r,0}, \dots, u_{r,m})$ be r elements in $\mathbb{Z}^{m+1}$, and let $v_1, \dots, v_r \in \mathbb{Z}$, such that for all $i \in \{0, \dots, m\}$, we get $\sum_{j=1}^{r} v_j u_j u_{j,i} = 0$.

Let $R \in E$, and fix arbitrary choices of cubical points $\widetilde{R}, \widetilde{R + P_i}$. Then the function

$$f : E \longrightarrow \mathbb{P}^1, \qquad R \longmapsto \prod_{j=1}^{r} Z_n(u_{j,0}\widetilde{R} + \sum_{i=1}^{m} u_{j,i}\widetilde{P}_i + \widetilde{T}_i)^{v_j},$$

is a well-defined function on E with divisor

$$\operatorname{div}(f) = \sum_{j=1}^{r} n v_j [u_{j,0}]^* (-\sum_{i=1}^{m} u_{j,i} P_i - T_i).$$

Here, $Z_n = Z_1^n$, and the cubical point $u_{j,0}\widetilde{R} + \sum_{i=1}^{m} u_{j,i}\widetilde{P}_i + \widetilde{T}_i$ is computed by combining the level-n cubical exponentiation with the cubical translation by $\widetilde{T}_i$.

Example 6. Let $P \in E[\ell]$, where $\ell = 2m$ is even. Working in level $n = 2$, the function

$$f_{\ell,P} : R \longmapsto \frac{Z_2(m\widetilde{P} + \widetilde{R} + \widetilde{P_0}) Z_2(\widetilde{R})^{m-1}}{Z_2(\widetilde{P + R})^m}$$

depends only on the choice of $\widetilde{P}$ and $\widetilde{P_0}$, where $P_0 = mP \in E[2]$, and its divisor is

$$\operatorname{div} f_{\ell,P} = 2(-mP - P_0) + 2(m-1)(0_E) - 2m(-P) = \ell(0_E) - \ell(-P).$$

A.10 Pairings in level 2

We obtain level-2 cubical pairings by replacing Z_1 with Z_2 in Theorem 3. Since $Z_2 = Z_1^2$, we obtain the square of the Weil and Tate pairings, yielding the results of Section 5.1. This square loses one bit of information when ℓ is even, but in that case one can use the cubical translation by appropriate points of 2-torsion to recover the true Weil and Tate pairings, see [Rob24, Theorem 2.9].

There is no need to use uniformizers as in Remark 8 for level 2: instead we can use the X coordinate, since for any $\widetilde{R}$ above 0_E we have $X(\widetilde{R}) \neq 0$. Indeed, in the Weil pairing we can replace $Z(\ell\widetilde{Q})/Z(\ell\widetilde{P})$ with $X(\ell\widetilde{Q})/X(\ell\widetilde{P})$, and in the Tate pairing we can replace $Z(\widetilde{0}_E)/Z(\ell\widetilde{P})$ with $X(\widetilde{0}_E)/X(\ell\widetilde{P})$. Moreover, recall that with our normalization $\widetilde{0}_E$ satisfies $X(\widetilde{0}_E) = 1$.

We now get the level-2 cubical pairing formulæ from Theorem 4, or Example 6:

Theorem 5. *Let $\ell = 2m$ be even.*

- *If $P, Q \in E[\ell]$, then their ℓ-Weil pairing is*

$$e_{W,\ell}(P,Q) = \frac{Z(m\widetilde{P} + \widetilde{Q} + \widetilde{P_0}) Z(m\widetilde{Q} + \widetilde{Q_0}) Z(\widetilde{P})}{Z(m\widetilde{P} + \widetilde{P_0}) Z(m\widetilde{Q} + \widetilde{P} + \widetilde{Q_0}) Z(\widetilde{Q})} = \frac{Z(m\widetilde{P} + \widetilde{Q} + \widetilde{P_0}) X(m\widetilde{Q} + \widetilde{Q_0}) Z(\widetilde{P})}{X(m\widetilde{P} + \widetilde{P_0}) Z(m\widetilde{Q} + \widetilde{P} + \widetilde{Q_0}) Z(\widetilde{Q})},$$

  where $\widetilde{P}, \widetilde{Q}, \widetilde{P + Q}, \widetilde{P_0}$, and $\widetilde{Q_0}$ are arbitrary cubical points over P, Q, $P + Q$, $P_0 := [m]P$, and $Q_0 := [m]Q$, respectively.

- *If $P \in E[\ell](\mathbb{F}_q)$ and $Q \in E(\mathbb{F}_q)$, then their (non-reduced) Tate pairing is*

$$e_{T,\ell}(P,Q) = \frac{Z(m\widetilde{P} + \widetilde{Q} + \widetilde{P_0}) Z(\widetilde{0}_E)}{Z(\widetilde{Q}) Z(m\widetilde{P} + \widetilde{P_0})} = \frac{Z(m\widetilde{P} + \widetilde{Q} + \widetilde{P_0}) X(\widetilde{0}_E)}{Z(\widetilde{Q}) X(m\widetilde{P} + \widetilde{P_0})},$$

where $\widetilde{P}$, $\widetilde{Q}$, and $\widetilde{P+Q}$ are arbitrary rational *cubical points above P, Q, and $P+Q$, respectively, such that $Z(\widetilde{P+Q})/Z(\widetilde{P})Z(\widetilde{Q})$ is a square in $\mathbb{F}_q$, and $\widetilde{P_0}$ is an arbitrary cubical point over $P_0 := [m]P$.*

Proof. We prove the formula for the Tate pairing. By Example 6, we have:

$$f_{\ell,P}((Q)-(0_E)) = \frac{Z(m\widetilde{P}+\widetilde{Q}+\widetilde{P_0})Z(\widetilde{Q})^{m-1}Z(\widetilde{P})^m}{Z(\widetilde{P+Q})^m Z(m\widetilde{P}+\widetilde{P_0})Z(\widetilde{0}_E)^{m-1}}$$

As before, to get the non-reduced Tate pairing and make the above equation well-defined, we multiply both sides by $(x/y)^\ell(0_E)$:

$$e_{T,\ell}(P,Q) = \frac{Z(m\widetilde{P}+\widetilde{Q}+\widetilde{P_0})X(\widetilde{0}_E)}{Z(\widetilde{Q})X(m\widetilde{P}+\widetilde{P_0})}\frac{Z(\widetilde{Q})^m Z(\widetilde{P})^m}{Z(\widetilde{P+Q})^m (Z/(x/y)^2)(\widetilde{0}_E)^m},$$

with $X(\widetilde{0}_E) = (Z/(x/y)^2)(\widetilde{0}_E) = 1$, and since by assumption $Z(\widetilde{P+Q})/Z(\widetilde{P})Z(\widetilde{Q})$ is a square in $\mathbb{F}_q$, $Z(\widetilde{Q})^m Z(\widetilde{P})^m / Z(\widetilde{P+Q})^m$ is an ℓ-th power in $\mathbb{F}_q$. □