Effective Pairings in Isogeny-based Cryptography

Krijn Reijnders

Radboud University, Nijmegen, The Netherlands
krijn@cs.ru.nl

Abstract. Pairings are useful tools in isogeny-based cryptography and have been used in SIDH/SIKE and other protocols. As a general technique, pairings can be used to move problems about points on curves to elements in finite fields. However, until now, their applicability was limited to curves over fields with primes of a specific shape and pairings seemed too costly for the type of primes that are nowadays often used in isogeny-based cryptography. We remove this roadblock by optimizing pairings for highly-composite degrees such as those encountered in CSIDH and SQISign. This makes the general technique viable again: We apply our low-cost pairing to problems of general interest, such as supersingularity verification and finding full-torsion points, and show that we can outperform current methods, in some cases up to four times faster than the state-of-the-art. Furthermore, we analyze how pairings can be used to improve deterministic and dummy-free CSIDH. Finally, we provide a constant-time implementation (in Rust) that shows the practicality of these algorithms.

Keywords: post-quantum cryptography, isogeny, pairings, CSIDH

1 Introduction

In the event that quantum computers break current cryptography, post-quantum cryptography will provide the primitives required for digital security. Isogeny-based cryptography is a field with promising quantum-secure schemes, offering small public keys in key exchange (CSIDH [10]), and small signatures (SQISign [19]). The main drawback of isogeny-based cryptography is speed, as it requires heavy mathematical machinery in comparison to other areas of post-quantum cryptography. In particular, to ensure security against real-world side-channel analysis, the requirements for constant-time and leakage-free implementations

[*] This work was done in large while the author was on an internship at the Cryptographic Research Centre of the Technology Innovation Institute, Abu Dhabi, UAE. In particular, the author thanks Francisco Rodríguez-Henríquez and Michael Scott for their warm support and excellent advice on the performance of these pairings.

Date of this document: 2023-08-07.

cause a significant slowdown. Trends in current research in isogenies are, therefore, looking at new ideas to improve constant-time performance [1, 2, 7, 11, 13, 14, 25, 36, 37, 39], and analyzing side-channel threats [3, 8, 9, 31].

Surprisingly, although pairings were initially considered in SIDH and SIKE to improve the cost of key compression [16, 17], they have received little attention for optimizing CSIDH or later isogeny-based protocols. There are clear obstructions that heavily affect the performance of pairings: we have no control over the Hamming weight of p for the base fields (in CSIDH or SQISign), we are likely to compute pairings of highly-composite degree, and many optimizations in the pairing-based literature require different curve models than the ones we consider in isogeny-based cryptography. Nevertheless, the field of pairing-based cryptography is rich in ideas and altogether many small improvements can make pairings efficient even for unpractical curves. As a general technique, we can use pairings to analyze certain properties of *points on elliptic curves* by a pairing evaluation as *elements of finite fields*. In this way, a single pairing can be used to solve a curve-theoretical problem with only field arithmetic, which is much more efficient. Hence, even a relatively expensive pairing computation can become cost-effective if the resulting problem is much faster to solve "in the field" than "on the curve".

Pairings have been used constructively since the 2000s [24, 26]. The literature is rich, but the main focus has mostly been on pairings of prime degree. Although proposals using composite degree pairings have been analyzed, analysis such as Guillevic [23] shows that prime degrees are favorable. Composite degree pairings have thus received little attention compared to prime degree pairings.

Apart from the issues mentioned, CSIDH is well-suited for pairings, as it mostly works with points of order $N \mid p+1$ on supersingular curves $E/\mathbb{F}_p$ whose x-coordinate lives in $\mathbb{F}_p$. This allows for fast x-only arithmetic on the Kummer line $\mathbb{P}(\mathbb{F}_p)$, but also implies an embedding degree of 2 for pairings of degree $N \mid p+1$. The result $\zeta = e_N(P,Q)$ of a pairing is thus a norm-1 element in $\mathbb{F}_{p^2}$, and ζ contains useful information on P and Q, which is precisely the information that CSIDH often requires to perform certain isogenies. Hence, pairings are a natural tool to study properties of the pair (P,Q). Norm-1 elements in $\mathbb{F}_{p^2}$ allow for very fast $\mathbb{F}_p$-arithmetic (compared to $\mathbb{F}_{p^2}$-arithmetic). A final advantage is that for any supersingular curve E_A we choose over $\mathbb{F}_p$, the result of a pairing always ends up in $\mathbb{F}_{p^2}$. This allows for techniques that require fixing some public value in $\mathbb{F}_{p^2}$ which would not generalize to curves; it would require fixing a value for *all* supersingular curves E_A, independent of $A \in \mathbb{F}_p$.

Our contributions. Our main contribution is combining an optimized pairing with highly-efficient arithmetic in $\mu_r \subseteq \mathbb{F}_{p^2}^*$ to solve isogeny problems faster. To achieve this, we first optimize the pairing and then apply this low-cost pairing to move specific problems from curves to finite fields. Specifically,

1. we optimize pairings on supersingular curves: in Miller's Algorithm, we first reduce the cost of subroutines Dbl and Add and then reduce the total number of subroutines using non-adjacent forms and windowing techniques.

2. we analyze the asymptotic and concrete cost of single and multi-pairings, in particular for supersingular curves over `p512` (the prime used in CSIDH-512).
3. we apply these low-cost pairings to develop alternative algorithms for supersingularity verification, verifying full-torsion points, and finding full-torsion points, using highly-efficient arithmetic available for pairing evaluations.
4. we discuss the natural role these algorithms have when designing 'real-world' isogeny-based protocols, in particular, CSIDH-variants that are deterministic and secure against side-channel attacks.
5. we provide a full implementation of most of these algorithms in Rust, following the "constant-time" paradigm, that shows such algorithms can immediately be used in practice to speed up deterministic variants of CSIDH. Our implementation is available at:

   https://github.com/Krijn-math/EPIC

Related work. This work can partly be viewed as a natural follow-up to [7, 11, 13], works that analyze CSIDH as a real-world protocol, that is, removing randomness and dummy operations. Independent work by Lin, Wang, Xu, and Zhao [32] applies pairings to improve the performance of SQISign [19]. This shows the potential of pairings in isogeny-based cryptography and we believe this work can contribute to improving performance even further.

Organization of the paper. Section 2 introduces the mathematical tools in pairing-based and isogeny-based cryptography required in the rest of the paper. Section 3 analyzes optimization of pairings to the setting used in isogeny-based cryptography, which allows us to apply pairings to optimize general problems in Section 4. In Section 5, we show that these pairing-based algorithms speed up current variants of deterministic, dummy-free CSIDH and can be used to construct ideas beyond those in use now.

2 Preliminaries

Notation. Throughout, p denotes a large prime used with the base field $\mathbb{F}_p$, and quadratic extension $\mathbb{F}_{p^2}$, realized as $\mathbb{F}_p(i)$ with $i^2 = -1$. Both ℓ and ℓ_i denote a small odd prime that divides $p+1$. A (supersingular) elliptic curve E_A is assumed to be in Montgomery form

$$E_A : y^2 = x^3 + Ax^2 + x, \quad A \in \mathbb{F}_p,$$

although our work also applies to other curve forms (most notably Edwards). μ_r denotes the set of r-th roots in $\mathbb{F}_{p^2}^*$. In particular, μ_{p+1} can be seen as $\mathbb{F}_{p^2}^*/\mathbb{F}_p^*$, the elements in $\mathbb{F}_{p^2}$ of norm 1, and μ_r for $r \mid p+1$ is a subgroup of μ_{p+1}.

Finite field operations are denoted as $\mathbf{M}$ for multiplications, $\mathbf{S}$ for squarings, and $\mathbf{A}$ for additions. Inversions ($\mathbf{I}$) and exponentiation ($\mathbf{E}$) are expressed in $\mathbf{M}$, $\mathbf{S}$ and $\mathbf{A}$ as far as possible. We use a cost model of $1\mathbf{S} = 0.8\mathbf{M}$ and $1\mathbf{A} = 0.01\mathbf{M}$ to compare performance in terms of finite field multiplications.

2.1 Isogeny-based cryptography

This work deals with specific problems in isogeny-based cryptography. We assume a basic familiarity with elliptic curve arithmetic, e.g. Montgomery ladders and addition chains. A great introduction is given by Costello and Smith [18].

The prime p. We specifically look at supersingular elliptic curves over $\mathbb{F}_p$, where $p = h \cdot \prod_{i=1}^{n} \ell_i - 1$, with h is a suitable cofactor and the ℓ_i are small odd primes. We refer to these ℓ_i as *Elkies primes* [10]. We denote the set of Elkies primes as L_χ[1] and write $\ell_\chi = \prod_{\ell_i \in L_\chi} \ell_i$. Hence, $\log p = \log h + \log \ell_\chi$. If h is large, the difference in bit-size between ℓ_χ and $p+1$ can be significant, and this can impact performance whenever an algorithm takes either $\log \ell_\chi$ or $\log p$ steps. For $\mathtt{p512}$, h is only 4 and so we do not differentiate between the two.

Torsion points. Let E be a supersingular elliptic curve over $\mathbb{F}_p$, then E has $p+1$ rational points. Such points $P \in E(\mathbb{F}_p)$ therefore have order $N \mid p+1$. When $\ell_i \mid N$, we say P has ℓ_i-torsion. When P is of order $p+1$, we say P is a *full-torsion* point. The twist of E over $\mathbb{F}_p$ is denoted by E^t, and E^t is also supersingular. Rational points of E^t can also be seen as $\mathbb{F}_{p^2}$-points in $E[p+1]$ of the form (x, iy) for $x, y \in \mathbb{F}_p$. Using x-only arithmetic, we can do arithmetic on *both* $E(\mathbb{F}_p)$ and $E^t(\mathbb{F}_p)$ using only these rational x-coordinates.

CSIDH. We briefly revisit CSIDH [10] to show where full-torsion points appear, and refer to [10, 37, 39] for more details. CSIDH applies the class group action of $\mathrm{Cl}(\mathcal{O})$ on supersingular elliptic curves E_A over $\mathbb{F}_p$ whose rational endomorphism ring $\mathrm{End}_p(E_A) \cong \mathcal{O}$ to create a non-interactive key exchange. Given a starting curve E_0, Alice's private key is an ideal class $[\mathfrak{a}] \in \mathrm{Cl}(\mathcal{O})$ and her public key is $E_A := \mathfrak{a} * E_0$, and equivalent for Bob with $[\mathfrak{b}]$ and $E_B := \mathfrak{b} * E_0$. Both can derive the shared secret $E_{AB} := \mathfrak{a} * E_B = \mathfrak{b} * E_A$, given only the other's public key. In reality, we cannot sample random ideal classes $[\mathfrak{a}] \in \mathrm{Cl}(\mathcal{O})$. Instead, we generate $\mathfrak{a}$ as a product of small ideals $\mathfrak{l}_i = (\ell_i, \pi - 1)$ and $\mathfrak{l}_i^{-1} = (\ell_i, \pi + 1)$ (the decomposition of (ℓ_i) into prime ideals), e.g., $\mathfrak{a} := \prod \mathfrak{l}_i^{e_i}$, where the e_i are secret.

Evaluating $\mathfrak{a} * E$ is done by the factorization of $\mathfrak{a}$ into $\mathfrak{l}_i$, where each $\mathfrak{l}_i^{\pm 1}$ can be evaluated using Vélu's formulas [48] if we have a point $P \in \ker(\pi \pm 1) \cap E[\ell_i]$. This requirement comes down to P being a rational point of order ℓ_i, with $\pi P = P$, hence $P \in E(\mathbb{F}_p)$ and $\mathfrak{l}_i^{+1} * E$ is evaluated as $E \to E/\langle P \rangle$, or $\pi P = -P$, hence P lives on the twist E^t and $\mathfrak{l}_i^{-1} * E$ is evaluated as $E^t \to E^t/\langle P \rangle$.

By sampling random points $P \in E(\mathbb{F}_p)$ and $Q \in E^t(\mathbb{F}_p)$ of order ℓ_i, we can use the right scalar multiple of either P or Q to compute the action of $\mathfrak{l}_i$ resp. $\mathfrak{l}_i^{-1}$ whenever ℓ_i divides $\mathrm{Ord}(P)$ resp. $\mathrm{Ord}(Q)$. The original points P and Q can then be pulled through the isogeny and used again as a new set of points on the codomain [10, 37, 39]. By repeating this procedure and sampling new points P, Q when necessary, we compute the full action of $\mathfrak{a} * E$. As P and Q are sampled randomly, they have probability $\frac{\ell_i - 1}{\ell_i}$ that ℓ_i divides their order.

[1] pronounced "ell-kie"

Deterministic CSIDH. The probabilistic nature of the evaluation of $\mathfrak{a} * E$, stemming from the random sampling of points P, Q, causes several issues, as randomness makes constant-time implementations difficult [1, 7, 13], leaks secret information through physical attacks [3], and requires a good source of entropy, which can be expensive or difficult on certain devices.

One way to avoid this random nature of $\mathfrak{a} * E$ is to ensure that both P and Q are *full-torsion points*, e.g., we have $\mathrm{Ord}(P) = \mathrm{Ord}(Q) = p+1$. For a point P that is not a full-torsion point, we say P *misses* some torsion ℓ_i and we denote the *missing torsion* for P by $\mathsf{Miss}(P)$. Note that $\mathsf{Miss}(P)\cdot\mathsf{Ord}(P) = p+1$. CSIDH strategies that require only two full-torsion points $T_+ \in E(\mathbb{F}_p)$ and $T_- \in E^t(\mathbb{F}_p)$ were discussed in [11] and implemented and improved by [7, 13]. These restrict to coefficients $e_i \in \{-1, +1\}$ to remove randomness and dummy operations.

2.2 Building blocks in isogeny-based cryptography.

We list several general routines in isogeny-based cryptography that will be analyzed in more detail in later sections. These routines are posed as general problems, with their role in CSIDH specified afterward.

1. Finding the order of a point: Given $P \in E(\mathbb{F}_p)$, find the order $\mathrm{Ord}(P)$.
2. Verifying supersingularity: Given $A \in \mathbb{F}_p$, verify E_A is supersingular.
3. Verifying full-torsion points: Given two points $P \in E(\mathbb{F}_p)$, $Q \in E^t(\mathbb{F}_p)$ verify P and Q are full-torsion points.
4. Finding full-torsion points: Given a curve E_A, find two full-torsion points $P \in E(\mathbb{F}_p)$ and $Q \in E^t(\mathbb{F}_p)$.

It is easy to see that these problems are related. For example, verifying supersingularity is usually done by verifying that E_A has order $p+1$, by showing that there is a $\mathbb{F}_p$-rational point of order $N \geq 4\sqrt{p}$. This implies $N \mid \#E_A(\mathbb{F}_p)$ and hence we must have $\#E(\mathbb{F}_p) = p+1$ as $p+1$ is the only possible remaining value in the Hasse interval.

All variants of CSIDH use a supersingularity verification in order to ensure a public key E_A is valid. dCSIDH [7] includes full-torsion points (P, Q) in the public key to speed up the shared-secret computation. This requires finding full-torsion points in key generation and, given the public key, verifying such points (P, Q) are full-torsion before deriving the shared secret. Without this verification, a user is vulnerable to side-channel attacks.

2.3 Pairing-based cryptography

One of the goals in pairing-based cryptography is to minimize the cost of computing a Weil or Tate pairing. We assume a basic familiarity with pairings up to the level of Costello's tutorial [15]. Other great resources are Galbraith [21] and Scott [43]. We focus only on the reduced Tate pairing, as it is more efficient for our purposes. We build on top of the fundamental works [4, 5, 38, 49].

The reduced Tate pairing. In this work, we are specifically focused on the reduced Tate pairing of degree r for supersingular elliptic curves with embedding degree $k = 2$, which can be seen as a bilinear pairing

$$e_r : E[r] \times E(\mathbb{F}_{p^2})/rE(\mathbb{F}_{p^2}) \to \mathbb{F}_{p^2}^*/(\mathbb{F}_{p^2}^*)^r.$$

In the *reduced* Tate pairing, the result $\zeta = e_r(P, Q)$ is raised to the power $k = (p^2 - 1)/r$, which ensures ζ^k is an r-th root of unity in μ_r. In this work, we want to evaluate the Tate pairing on points $P \in E(\mathbb{F}_p)$ and $Q \in E^t(\mathbb{F}_p)$ of order r. For supersingular curves over $\mathbb{F}_p$ and $r \mid p+1$, such points generate all of $E[r]$. From the point of view of pairings, $E(\mathbb{F}_p)$ is the *base-field subgroup* and $E^t(\mathbb{F}_p)$ is the *trace-zero subgroup* of $E[p+1]$. Using the bilinear properties of the Tate pairing, we can compute $e_r(P, Q)$ from its restriction to $E(\mathbb{F}_p) \times E^t(\mathbb{F}_p)$.

Computing the Tate pairing. There are multiple ways to compute the Tate pairing [33, 34, 38, 46]. Most implementations evaluate e_r in three steps.

1. compute the Miller function f_{rP}, satisfying $\mathrm{div}(f_{rP}) = r(P) - r(\mathcal{O})$,
2. evaluate f_{rP} on an appropriate divisor D_Q,
3. raise $f_{rP}(D_Q)$ to the appropriate power, $\frac{p^2-1}{r}$, i.e., $e_r(P, Q) = f_{rP}(D_Q)^{p-1}$.

In practice, f_{rP} is a function in x and y of degree r, where r is cryptographically large, and therefore infeasible to store or evaluate. Miller's solution is a bitwise computation and direct evaluation of f_{rP} on D_Q to compute $f_{rP}(D_Q)$ in $\log(r)$ steps. By the work of Barreto, Kim, Lynn, and Scott [4, Theorem 1], we are in the fortunate situation that we can choose $D_Q = Q$. The Hamming weight of r is a large factor in the cost of computing $f_{rP}(Q)$ as a single step in Miller's loop takes close to twice the computational cost if the bit is 1. For our purposes, $r = p+1$ or $r = \ell_\chi$, and thus we have little control over the Hamming weight of r. The last step is also known as *the final exponentiation*. Algorithm 1 describes Miller's Algorithm before applying any optimizations, where $l_{T,T}$ and $l_{T,P}$ denote the required line functions (see [15, § 5.3]). We refer to the specific subroutines in Line 3 as Dbl and Line 5 as Add.

Algorithm 1 Miller's Algorithm

```
Input: $P \in E(\mathbb{F}_p)$, $Q \in E^t(\mathbb{F}_p)$, $r$ of embedding degree $k = 2$, with $r = \sum_{i=0}^{t} t_i \cdot 2^i$
Output: The reduced Tate pairing $e_r(P, Q) \in \mu_r$
1: $T \leftarrow P$, $f \leftarrow 1$
2: for $i$ from $t - 1$ to 0 do
3:     $T \leftarrow 2T$, $f \leftarrow f^2 \cdot l_{T,T}(Q)$                // Dbl
4:     if $t_i = 1$ then
5:         $T \leftarrow T + P$, $f \leftarrow f \cdot l_{T,P}(Q)$        // Add
6: return $f^{p-1}$
```

More generally, the value f is updated according to the formula

$$f_{(n+m)P} = f_{nP} \cdot f_{mP} \cdot \frac{l}{v} \tag{1}$$

where l and v are the lines that arise in the addition of nP and mP. Miller's Algorithm uses $n = m$ to double $T = nP$, or $m = 1$ to add $T = nP$ and P.

2.4 Field arithmetic

The result of the reduced Tate pairing is a value $\zeta \in \mu_r \subseteq \mathbb{F}_{p^2}^*$ of norm 1, as it is an r-th root of unity. We require two useful algorithms from finite field arithmetic: Lucas exponentiation and Gauss's Algorithm to find primitive roots.

Gauss's algorithm. An algorithm attributed to Gauss [35, p. 38] to find primitive roots of a certain order in a finite field is given in Algorithm 2, specialized to the case of finding a generator α for a finite field $\mathbb{F}_q$. It assumes a subroutine Ord computing the order of any element in the finite field.

Algorithm 2 Gauss's Algorithm.

Input: A prime power $q = p^k$.
Output: A generator α for $\mathbb{F}_q^*$.

```
1: $\alpha \xleftarrow{\$} \mathbb{F}_q^*$, $t \leftarrow \mathsf{Ord}(\alpha)$
2: while $t \neq q - 1$ do
3:     $\beta \xleftarrow{\$} \mathbb{F}_q^*$, $s \leftarrow \mathsf{Ord}(\beta)$
4:     if $s = q - 1$ then return $\beta$
5:     else
6:         Find $d \mid t$ and $e \mid s$ with $\gcd(d, e) = 1$ and $d \cdot e = \mathrm{lcm}(t, s)$
7:         Set $\alpha \leftarrow \alpha^{t/d} \cdot \beta^{s/e}$, $t \leftarrow d \cdot e$
8: return $\alpha$
```

Gauss's Algorithm is easy to implement and finds generators quickly. The main cost is computing the orders. We can adapt Gauss's Algorithm to elliptic curves to find generators for $E(\mathbb{F}_p)$, simply by replacing the rôles of α, β by rational points P, P' until P reaches $\mathsf{Ord}(P) = p + 1$. Intuitively, one could say we "add" the torsion that P is missing using the right multiple of P'.

Lucas exponentiation. Lucas exponentiation provides fast exponentiation for $\zeta \in \mathbb{F}_{q^2}$ of norm 1. They are used in cryptography since 1996 [27, 28], and specifically applied to pairings by Scott and Barreto [44]. We follow their notation.

Let $\zeta = a + bi \in \mathbb{F}_{p^2}$ be an element of norm 1, i.e. $a^2 + b^2 = 1$, then ζ^k can be efficiently computed using only $a \in \mathbb{F}_p$ for every $k \in \mathbb{N}$ using Lucas sequences, based on simple laddering algorithms. We denote these sequences by $V_k(a)$ and $U_k(a)$ but often drop a for clarity. The central observation is

$$\zeta^k = (a+bi)^k = V_k(2a)/2 + U_k(2a) \cdot bi, \quad \text{for } \zeta \in \mu_{p+1},$$

where

$$\begin{aligned} V_0 = 2, \quad V_1 = a, \quad V_{k+1} = a \cdot V_k - V_{k-1}, \\ U_0 = 0, \quad U_1 = 1, \quad U_{k+1} = a \cdot U_k - U_{k-1}. \end{aligned}$$

Given V_k, we can compute U_k by $(a \cdot V_k - 2 \cdot V_{k-1})/(a^2 - 4)$. An algorithmic description is given in [44, App. A]. For this work, we only require the value of $V_k(2a)$. As such, exponentiation of norm-1 elements is much more efficient than general exponentiation in $\mathbb{F}_{p^2}^*$: the former requires $1\mathbf{S} + 1\mathbf{M}$ per bit of k, whereas the latter requires roughly $2\mathbf{S} + \frac{5}{2}\mathbf{M}$ per bit of k, assuming the Hamming weight of k is $\log(k)/2$. In our cost model, this is an almost 60% improvement. We denote the cost of exponentiation per bit for norm-1 elements by C_{Lucas}.

This arithmetic speed-up is key to the applications in this work: as the required pairings have evaluations of norm 1, we can apply Lucas exponentiation to the results to get very fast arithmetic. In comparison to x-only arithmetic on the curve, we are between five and six times faster per bit. Hence, if the cost of the pairing is low enough, the difference in cost between curve arithmetic and Lucas exponentiation is so large that it makes up for the cost of the pairing.

3 Optimizing pairings for composite order

In this section, we apply several techniques to decrease the cost of Miller's Algorithm, specifically for pairings of degree $r \mid p+1$, and points $P \in E(\mathbb{F}_p)$, $Q \in E^t(\mathbb{F}_p)$, with E supersingular and $p = h \cdot \ell_\chi - 1$. This is a different scenario than pairing-based literature usually considers: we have no control over the Hamming weight of $p+1$, and we compute pairings of composite degree.

We first give an abstract view and then start optimizing Miller's Algorithm. In Section 3.2, we decrease the cost per subroutine Dbl/Add with known optimizations that fit our scenario perfectly. In Section 3.3, we decrease the number of subroutines Dbl/Add using non-adjacent forms and (sliding) window techniques, inspired by finite field exponentiation and elliptic curves scalar multiplication.

3.1 An abstract view on pairings

Silverman [45], views the reduced Tate pairing as a threefold composition

$$E[r] \to \mathrm{Hom}(E[r], \mu_r) \to \mathbb{F}_{p^2}^*/\mathbb{F}_{p^2}^{*,r} \xrightarrow{z \mapsto z^{(p^2-1)/r}} \mu_r(\mathbb{F}_{p^2}) \qquad (2)$$

similar to the one described in Section 2.3. Namely, for $r = p+1$, we can reduce the first map to $E(\mathbb{F}_p) \to \mathrm{Hom}(E^t(\mathbb{F}_p), \mu_r)$ to get

$$\Psi : E(\mathbb{F}_p) \to \mathrm{Hom}(E^t(\mathbb{F}_p), \mu_r), \quad P \mapsto e_r(P, -),$$

which can be made concrete as the Miller function $P \mapsto f_{rP}$. By composing with its evaluation on Q, we get $f_{rP}(Q) = e_r(P, Q)$ (unreduced). To $f_{rP}(Q)$, we apply the final exponentiation $z \mapsto z^{(p^2-1)/r}$. In the case of $r = p + 1$, we thus get the reduced Tate pairing $e_{p+1}(P, Q)$ as $\zeta = f_{(p+1)P}(Q)^{p-1}$.

From this point of view, identifying full-torsion points $P \in E(\mathbb{F}_p)$ is equivalent to finding points P that map to isomorphisms $f_{rP} \in \mathrm{Hom}(E^t(\mathbb{F}_p), \mu_r)$. We make this precise in the following lemma.

Lemma 1. *Let E be a supersingular curve over $\mathbb{F}_p$. Let $P \in E(\mathbb{F}_p)$ and $r = p+1$. Then f_{rP} as a function $E^t(\mathbb{F}_p) \to \mu_r$ has kernel*

$$\ker f_{rP} = \{Q \in E^t(\mathbb{F}_p) \mid \mathsf{Ord}(P) \text{ divides } \mathsf{Miss}(Q)\}.$$

Hence, $|\ker f_{rP}| = \mathsf{Miss}(P)$. Thus, if P generates $E(\mathbb{F}_p)$, the kernel is trivial.

Proof. Recall that $\mathsf{Ord}(P) \cdot \mathsf{Miss}(P) = p + 1$. The reduced Tate pairing maps precisely to an exact $p + 1$-th root of unity if and only if P and Q are a torsion basis for $E[p + 1]$ [45]. Note that if $P \in E(\mathbb{F}_p)$ does not have order $p + 1$, we can write $P = [\mathsf{Miss}(P)]T_+$ for some specific point $T_+ \in E(\mathbb{F}_p)$ oforder $p + 1$, and similarly $Q = [\mathsf{Miss}(Q)]T_-$ for some $T_- \in E^t(\mathbb{F}_p)$. Hence $e_{p+1}(T_+, T_-)$ is an exact $p + 1$-th root of unity, and

$$\zeta = e_{p+1}(P, Q) = e_{p+1}(T_+, T_-)^{\mathsf{Miss}(P) \cdot \mathsf{Miss}(Q)}$$

is a $p + 1$-th root of unity with $\mathsf{Miss}(\zeta) = \mathrm{lcm}(\mathsf{Miss}(P), \mathsf{Miss}(Q))$. Whenever $\mathsf{Ord}(P)$ divides $\mathsf{Miss}(Q)$, we know that $p + 1$ divides $\mathsf{Miss}(P) \cdot \mathsf{Miss}(Q)$ and so we must have $\mathrm{lcm}(\mathsf{Miss}(P), \mathsf{Miss}(Q)) = p + 1$, i.e. $\zeta = 1$. As $\mathsf{Ord}(P) \mid \mathsf{Miss}(Q)$ implies $\mathsf{Ord}(Q) \mid \mathsf{Miss}(P)$, we can generate all such points Q by a single point R of order $\mathsf{Miss}(P)$, giving us $\ker f_{rP} = \langle R \rangle$ of order $\mathsf{Miss}(P)$. $\square$

Note that, given a full-torsion point $P \in E(\mathbb{F}_p)$, we can thus identify full-torsion points $Q \in E^t(\mathbb{F}_p)$ as points where $e_r(P, Q)$ is a primitive root in μ_r. In light of Lemma 1, we can try to tackle the routines sketched in Section 2.2 using properties of f_{rP}, $f_{rP}(Q)$ and $\zeta = f_{rP}(Q)^{p-1}$. For example, we can find $\ker f_{rP}$ by evaluating f_{rp} on multiple points Q_i, and finding the orders of the resulting elements ζ_i. In the language of pairing-based cryptography, we compute multiple pairings $e_r(P, Q_i)$ for the same point P. Hence, we need to minimize the cost of several evaluations of the Tate pairing for fixed P but different points Q_i.

3.2 Reducing the cost per subroutine of Miller's loop

We now optimize the cost per Dbl and Add in Miller's Algorithm. We assume that $P \in E(\mathbb{F}_p)$ is given by $\mathbb{F}_p$-coordinates $x_P, y_P \in \mathbb{F}_p$, and $Q \in E^t(\mathbb{F}_p)$ can be given by $\mathbb{F}_p$-coordinates $x_Q, y_Q \in \mathbb{F}_p$ (we implicitly think of Q as $(x_Q, i \cdot y_Q)$).

Some of these techniques were used before in SIDH and SIKE [16, 17], in a different situation: in SIDH and SIKE, these pairings were specifically applied for $p = 2^{e_2} \cdot 3^{e_3} - 1$, whereas we assume $p = h \cdot \ell_\chi - 1$. Thus, we have much more different $\ell_i \mid p + 1$ to work with, and we cannot apply most of their techniques.

Representations. For T, we use projective coordinates to avoid costly inversions when doubling T, adding P and computing $\ell_{T,T}$ and $\ell_{T,P}$. For Q, as we only evaluate Q in $\ell_{T,T}$ and $\ell_{T,P}$, we leave Q affine. $f = a + bi$ is an $\mathbb{F}_{p^2}$-value represented projectively as $(a : b : c)$, with $a, b, c \in \mathbb{F}_p$ and c as the denominator. Although x-only pairings exist [22], they seem unfit for this specific scenario.

The final exponentiation. As established before, after computing $f_{rP}(Q) \in \mathbb{F}_{p^2}$ we perform a final exponentiation by $p-1$. This is beneficial for two reasons: **1.)** Raising to the power p is precisely applying Frobenius $\pi : z \mapsto z^p$ in $\mathbb{F}_{p^2}$, and so $\pi : a + bi \mapsto a - bi$. Hence we can compute $z \mapsto z^{p-1}$ as $z \mapsto \frac{\pi(z)}{z}$. The Frobenius part is 'free' in terms of computational cost. In $\mathbb{F}_{p^2}$, $z \mapsto z^{-1}$ is simply $(a + bi)^{-1} = \frac{(a-bi)}{a^2+b^2}$. Hence, the $\mathbb{F}_p$-inversion of $a^2 + b^2$ is the dominating cost of the final exponentiation. In constant-time, this costs about $\log p$ multiplications. When a and b are public, we use faster non-constant-time inversions. In comparison to prime pairings, this final exponentiation is surprisingly efficient. **2.)** Raising z to the power $p-1$ for $\mathbb{F}_{p^2}$-values gives the same as $\alpha \cdot z$ for $\alpha \in \mathbb{F}_p^*$, as $(\alpha \cdot z)^{p-1} = \alpha^{p-1} \cdot z^{p-1} = z^{p-1}$. Hence, when we compute $f_{rP}(Q) \in \mathbb{F}_{p^2}$, we can ignore or multiply by $\mathbb{F}_p$-values [4], named "denominator elimination" in pairing literature. That is, we ignore the denominator c in the representation $(a : b : c)$ of f in the Miller loop, and similarly in evaluating $l_{T,T}(Q)$ and $l_{T,P}(Q)$, saving several $\mathbb{F}_p$-operations in Dbl/Add [12, 41].

Reusing intermediate values. In Dbl, computing $T \leftarrow 2T$ shares many values with the computation of $l_{T,T}$ and in Add computing $T \leftarrow T + P$ shares values with $l_{T,P}$. Reusing such values saves again several $\mathbb{F}_p$-operations in Dbl/Add.

Improved doubling formulas. As shown in [17, §4.1], the subroutine Dbl in Miller's Algorithm is more efficient when using a projective representation of $T \in E(\mathbb{F}_p)$ as (X^2, XZ, Z^2, YZ), although this requires a slight adjustment of the formulas used in Add. Overall, this reduces the cost for Dbl to $5\mathbf{S} + 15\mathbf{M}$ and the cost for Add becomes $4\mathbf{S} + 20\mathbf{M}$, for an average of $\mathbf{7S} + 25\mathbf{M}$ per bit.

3.3 Reducing the number of subroutines in the Miller loop

Next to reducing the cost for a single Dbl/Add, we apply techniques to reduce the total number of Adds. Usually in pairing-based cryptography, we do so by using primes p of low Hamming weight. Here we do not have this freedom, thus we resort to techniques inspired by exponentiation in finite fields.

Non-Adjacent Form. With no control over the Hamming weight of $p + 1$, we assume half of the bits are 1. However, in Miller's Algorithm, it is as easy to add $T \leftarrow T + P$ as it is to subtract $T \leftarrow T - P$ (which we denote Sub), with the only difference being a negation of y_P. Hence, we use *non-adjacent forms*

(NAFs [40]) to reduce the number of Add/Subs. A NAF representation of $p+1$ as

$$p+1=\sum_{i=0}^{n} t_i\cdot 2^i, \quad t_i\in\{-1,0,1\},$$

reduces the Hamming weight from $\log(p)/2$ to $\log(p)/3$, and thus decrease the number of expensive Add/Subs in Miller's Algorithm by $\log(p)/6$. We get an average cost of $6\frac{1}{3}\mathbf{S}+21\frac{2}{3}\mathbf{M}$ per bit, a saving of about 10%.

Algorithm 3 gives a high-level overview of the Miller loop with the improvements so far, for general p. Note that, as the output $\zeta\in\mathbb{F}_{p^2}$ will have norm 1, we only require the real part of ζ. See Appendix A for the specific algorithms Dbl, Add and Sub, which are implemented in `bignafmiller.rs`.

Algorithm 3 Miller's algorithm, using NAFs

```
Input: $x_P, y_p, x_Q, y_Q \in \mathbb{F}_p$, $p+1=\sum_{i=0}^{t} t_i\cdot 2^i$
Output: The real part of $e_r(P,Q)\in\mu_{p+1}$
 1: $T=(X^2, XZ, Z^2, YZ)\leftarrow(x_P^2, x_P, 1, y_P)$
 2: $f\leftarrow(1,0)$
 3: for $i$ from $t-1$ to 0 do
 4:     $(T,f)\leftarrow \mathsf{Dbl}(T,f,x_Q,y_Q)$
 5:     if $t_i=1$ then
 6:         $(T,f)\leftarrow \mathsf{Add}(T,f,x_P,y_P,x_Q,y_Q)$
 7:     if $t_i=-1$ then
 8:         $(T,f)\leftarrow \mathsf{Sub}(T,f,x_P,y_P,x_Q,y_Q)$
 9: $a\leftarrow f[0]$, $b\leftarrow f[1]$
10: $\zeta\leftarrow\frac{a^2-b^2}{a^2+b^2}$                // The final exponentiation
11: return $\zeta$
```

For specific primes. For specific primes p, such as `p512`, we can improve on the NAF representation by using windowing techniques [29, 30]. This allows us to decrease even further the times we need to perform Add or Sub, at the cost of a precomputation of several values.

In short, windowing techniques allow us to not only add or subtract P but also multiples of P during the loop. To do so, we are required to precompute several values, namely $iP, -iP, f_{iP}$ and f_{-iP}. We need the multiples $\pm iP$ to perform $T\leftarrow T\pm iP$, and the line values f_{iP} to set $f\leftarrow f\cdot f_{\pm iP}\cdot\ell_{T,iP}(Q)$ in Add/Sub. We first precompute the required iP in projective form, and we keep track of f_{iP}. We use Montgomery's trick to return the points iP in affine form at the cost of a single inversion and some multiplications. Using affine form decreases the cost of $T\leftarrow T\pm iP$ during Add/Sub.

We note that iP gives $-iP$ for free, simply by negating y_{iP}. Furthermore, from $f_{iP}=a+bi$ we can obtain f_{-iP} as $f_{iP}^{-1}=\frac{a-bi}{a^2+b^2}$. However, as $a^2+b^2\in\mathbb{F}_p^*$, we can ignore these (thanks to the final exponentiation) and simply set $f_{-iP}=a-bi$.

Altogether, these sliding-window techniques reduce the number of Add/Subs from $\log(p)/3$ down to about $\log(p)/(w+1)$. See `bigwindowmiller.rs` for the implementation of this algorithm.

For the prime p512, we found the optimum at a window size $w = 5$. This requires a precomputation of $\{P, 3P, \ldots, 21P\}$. Beyond $w = 5$, the cost of additional computation does not outweigh the decrease in Add/Subs. Altogether this gives another saving of close to 10% for this prime.

Remark 1. A reader who is familiar with curve arithmetic might be tempted to suggest (differential) addition chains at this point, specialized for $p + 1$, to decrease even further the number of Add/Subs. However, an addition chain requires either keeping track of the different points in the chain in projective form, hence performing projective additions/subtractions, or, converting them to affine points and performing affine additions/subtractions. Both approaches are not cost-effective; the amount of additions/subtractions hardly decreases, and the cost per addition/subtraction increases significantly. The crucial difference is that the precomputed points can be mapped into affine form using a single shared inversion and a few multiplications.

3.4 Multiple pairing evaluations.

The previous sections attempt to optimize a single pairing $e_r(P, Q)$. However, in many scenarios, including the ones in Section 4, it is beneficial to optimize the cost of multiple pairings, in particular multiple pairings $e_r(P, Q_1), \ldots e_r(P, Q_k)$ for the same point P. This is known as the "one more pairing" problem in pairing literature. We quickly sketch two methods to do so. Firstly, assuming the set $\{Q_1, \ldots, Q_k\}$ is known in advance and we minimize the overall cost of these k pairings. Secondly, assuming we want to compute additional pairings $e_r(P, Q_{k+1})$ after having already computed $e_r(P, Q_1), \ldots, e_r(P, Q_k)$.

Evaluating a fixed set $\boldsymbol{Q_1, Q_2, \ldots, Q_k}$. Optimizing k pairings $e_r(P, Q_i)$ for an already known set $Q_1, \ldots, Q_k$ is easy: only f depends on Q_i, hence we can easily adapt Algorithm 3 for an array of points $[Q_1, \ldots, Q_k]$ to keep track of a value $f^{(i)}$ per Q_i, and we return an array $\zeta^{(1)}, \ldots, \zeta^{(k)}$. All evaluations share (per bit) the computations of T, $l_{T,T}$ and $l_{T,P}$. Our additional cost per extra point Q_i thus comes down to the evaluations $l_{T,T}(Q_i)$, $l_{T,P}(Q_i)$ and updating $f^{(i)}$. In total, this is $7\mathbf{M}$ per Dbl, and $5\mathbf{M}$ per Add/Sub, plus $2\mathbf{S} + \mathbf{I}$ to compute $\zeta^{(i)}$ given $f^{(i)}$ (the final exponentiation) per point Q_i. See `bigmultimiller.rs` for the implementation of this algorithm.

Evaluating additional points $\boldsymbol{Q_1, Q_2, \ldots}$ It is more difficult when we want to compute $e_r(P, Q')$ *after* the computation of $e_r(P, Q)$. I.e., in some applications we compute $e_r(P, Q)$ first, and, based on this evaluation, compute another $e_r(P, Q')$. In practice, this seems to require another full pairing computation.

Scott [42] observed that one can achieve a time/memory trade-off, by dividing Miller's Algorithm into three distinct subalgorithms: one to compute f_{rP}, one to evaluate $f_{rP}(Q_i)$ per Q_i and one for the final exponentiation. Paradoxically, this brings us back to the original three-step process from Section 2.3, where we argued that the degree of f_{rP} is too large to store f_{rP} in full. However, Scott notes, $f_{rP}(Q)$ can be computed from the set of all line functions $l_{T,T}$ and $l_{T,P}$ and Q. Up to $\mathbb{F}_p$-invariance, all such line functions l can be written as

$$l(x,y) = \lambda_x \cdot x + \lambda_y \cdot y + \lambda_0, \quad \lambda_i \in \mathbb{F}_p,$$

and we get a line function per Dbl, Add, and Sub. Thus, at a memory cost of

$$\underbrace{(\log(p) + 1/3\log(p))}_{\#\text{ subroutines}} \cdot \underbrace{3 \cdot \log(p)}_{\text{bits per } l} = 4 \cdot \log(p)^2$$

(the factor 1/3 can be decreased using windowing) we can store a representation of f_{rP} as an array of line functions. Hence, we can split up Algorithm 3 into three subroutines Construct, Evaluate, and Exponentiate, which coincide precisely with the decomposition of the Tate pairing given in Equation (2). We refer to the composition of these subalgorithms as Scott-Miller's algorithm. See Appendix B for an algorithmic description.

3.5 Summary of costs

We summarize Section 3 in terms of $\mathbb{F}_p$-operations for pairings of degree $r = p+1$.

General primes. Miller's Algorithm has $\log p$ steps and each step performs 1 Dbl. Using the techniques from Section 3.3 decreases the number of Add/Subs[2]:

1. each Dbl costs $15\mathbf{M} + 5\mathbf{S} + 7\mathbf{A}$, we always perform $\log p$ Dbls
2. each Add or Sub costs $20\mathbf{M} + 4\mathbf{S} + 9\mathbf{A}$,
   (a) in a naïve approach, we perform $\frac{1}{2}\log p$ Adds and Subs
   (b) using NAFs, we perform $\frac{1}{3}\log p$ Adds and Subs
   (c) using windowing, we perform $\frac{1}{w+1}\log p$ Adds and Subs

For CSIDH-512. For p512, Table 1 gives the number of $\mathbb{F}_p$-operations to compute a pairing, and shows the effectiveness of the optimizations: a reduction of 40% compared to unoptimized pairings.

For an additional pairing, if the points are known beforehand, we require only slightly more memory cost for each additional pairing. If we need to compute a multipairing for variable points, this takes $4 \cdot \log(p)^2$ bits of memory to store the representation of f_{rP}, using Scott-Miller's algorithm (Section 3.4).

[2] These techniques are inspired by finite field exponentiation and scalar multiplication, and have been analyzed previously for pairing-based cryptography [29, 30].

	M	**S**	**A**	Total
Original Miller's Algorithm	28 498	2621	39 207	30 987
Optimized step (Sec 3.2)	12 740	3569	12 230	15 717
Using NAFs (Alg. 3)	11 152	3254	11 125	13 866
Using windows ($w = 5$)	9963	2960	10 592	12 436
Additional pairing	4410	2	5704	4468

Table 1. Concrete cost of the Miller loop for `p512` for a pairing of degree $r = p + 1$. 'Total' gives the number of $\mathbb{F}_p$-operations, with cost model $1\mathbf{S} = 0.8\mathbf{M}$ and $1\mathbf{A} = 0.01\mathbf{M}$.

Remark 2. In Table 1, we consider the cost for a pairing of degree $r = p+1$. An alternative for primes $p = h \cdot \ell_\chi - 1$ with large cofactor h is to consider pairings of degree $r = \ell_\chi$ on $E[\ell_\chi]$. In many applications, such as [7], this contains all the necessary torsion information needed. The loop, then, has $\log \ell_\chi = \log p - \log h$ steps. The cost of such a pairing can be deduced from the given estimates.

4 Applications of pairings to isogeny problems

In this section, we apply the optimized pairing from Section 3 to the isogeny problems described in Section 2.1. The core design idea is clear: the pairing is now cheap enough to move isogeny problems from curves to finite fields, where we have highly efficient Lucas exponentiation. Lemma 1 captures what information about the original points remains after the pairing.

4.1 Verification of full-torsion points.

We start with the following problem: Given $P \in E(\mathbb{F}_p)$ and $Q \in E^t(\mathbb{F}_p)$, verify both points have full-torsion, for a supersingular curve E over $\mathbb{F}_p$ [3].

Current methods. Current methods to verify full-torsion points compute $[\frac{p+1}{\ell_i}]P \neq \mathcal{O}$ to conclude p has ℓ_i-torsion for every $\ell_i \mid p+1$ and hence is a full-torsion point. In a naïve way, this can be done per ℓ_i for the cost of a scalar multiplication of size $\log p$, using either Montgomery ladders or differential addition chains, at a total complexity of $\mathcal{O}(n \log p)$. Concretely, this comes down to a cost of $C_{\text{curve}} \cdot n \log p$ where C_{curve} is the cost of curve arithmetic per bit.

Using product trees this drops down to $\mathcal{O}(\log n \log p)$, although taking $\mathcal{O}(\log n)$ space. Product-tree-based order verification is currently the method used in state-of-the-art deterministic and dummy-free implementations [7] to verify a given basis (P, Q). Hence, doing this for both P and Q comes down to approximately $2 \cdot C_{\text{curve}} \cdot \log n \log p$ operations in $\mathbb{F}_p$.

[3] We treat *finding* full-torsion points in Section 4.3.

Torsion bases. We can improve on the previous verification matter by two easy observations. Firstly, whenever a pair of points $P \in E(\mathbb{F}_p)$ and $Q \in E^t(\mathbb{F}_p)$ generates all of $E[p+1] \subseteq E(\mathbb{F}_{p^2})$, the pair (P, Q) is a torsion basis for the $(p+1)$-torsion. As proposed in SIDH/SIKE [16, 17], we can verify such a torsion basis using the result that $\zeta = e_{p+1}(P, Q) \in \mu_{p+1}$ must be a $(p+1)$-th primitive root in $\mathbb{F}_{p^2}$. Secondly, this situation is ideal for our pairing: P has both rational coordinates, and Q has rational x and purely imaginary y-coordinate. As noted before, ζ is an element of norm 1 in $\mathbb{F}_{p^2}$, which allows us to apply fast Lucas exponentiation to compute ζ^k. The following lemma is our key building block.

Lemma 2. *Let $P \in E(\mathbb{F}_p)$ and $Q \in E^t(\mathbb{F}_p)$. Let $\zeta = e_{p+1}(P, Q) \in \mu_{p+1}$. Then*

$$\zeta^{\frac{p+1}{\ell_i}} \neq 1 \Leftrightarrow [\frac{p+1}{\ell_i}]P \neq \mathcal{O} \textit{ and } [\frac{p+1}{\ell_i}]Q \neq \mathcal{O}.$$

Proof. This is a direct application of Lemma 1. □

Hence, instead of verifying that both P and Q have ℓ_i-torsion, we verify that the powers $\zeta^{\frac{p+1}{\ell_i}}$ do not vanish. Furthermore, as ζ and its powers have norm 1, we simply verify that $\mathrm{Re}(\zeta^k) \neq 1$ which implies $\zeta^k \neq 1$. In terms of Lucas sequences, this is equal to $V_k(2a) \neq 2$ for $\zeta = a + bi$. Per bit, Lucas exponentiation is much more efficient than curve arithmetic, which allows us to compute every $\zeta^{\frac{p+1}{\ell_i}}$ (again using product trees) very efficiently at a similar complexity $\mathcal{O}(\log n \log p)$, and a concrete cost of $1 \cdot C_{\mathrm{Lucas}} \cdot \log n \log p$ [4].

Algorithm 4 summarizes this approach, where Order is a product-tree based function that computes the order of ζ (equivalently, verifies for which ℓ_i we have $\zeta^{\frac{p+1}{\ell_i}} \neq 1$) using Lucas exponentiation. Order is the field analogue of algorithms such as OrderRec [2] and validate_rec in CTIDH [1]. See `bigpairingfo.rs` for an implementation of Algorithm 4.

Algorithm 4 Verification of torsion basis

```
Input: $P \in E(\mathbb{F}_p)$, $Q \in E^t(\mathbb{F}_p)$
Output: True or False
1: $\zeta \leftarrow \mathrm{Re}(e_{p+1}(P, Q))$
2: $m \leftarrow \mathsf{Order}(\zeta)$
3: if $m = p + 1$ then
4:     return True
5: else
6:     return False
```

[4] To compute using Lucas exponentiation we use a constant-time laddering approach. Interesting future work would be to use (differential) addition chains to reduce costs.

Further improvements. As stated before, working in μ_{p+1} has the added benefit that we work in same subgroup of $\mathbb{F}_{p^2}$, independent of the curve E_A. This can be used to speed up the cost of torsion basis verification even more, at the cost of $\log p$ additional bits next to the pair (P, Q), as follows.

The scalars $\Lambda := \mathbb{Z}_{p+1}^*$ of invertible elements in $\mathbb{Z}_{p+1}$ act faithfully and free on both the group of full-torsion points of E_A, as well as on exact primitive roots in $\mu_{p+1} \subseteq \mathbb{F}_{p^2}$. This means that we can write any full-torsion point T relative to another full-torsion point T' as $T = [\lambda]T'$ with $\lambda \in \Lambda$. Similarly for primitive roots, this implies we can pick a system parameter ζ_0 as the "standard" primitive root, and can find $\lambda \in \Lambda$ such that for $\zeta = e_{p+1}(P, Q)$ we get $\zeta = \zeta_0^\lambda$.

By including λ next to (P, Q), we do not have to verify the complete order of ζ. Instead, we simply verify that $\lambda \in \Lambda$, compute $\zeta = e_{p+1}(P, Q)$ and verify that $\zeta = \zeta_0^\lambda$ which implies that ζ is an exact $(p + 1)$-th root of unity. Compared to the algorithm sketched before this means that instead of $\mathcal{O}(\log n \log p)$ to verify the order of ζ, we use a single Lucas exponentiation $\mathcal{O}(\log p)$ to verify ζ.

Remark 3. The addition of the discrete log λ such that $\zeta = e_{p+1}(P, Q) = \zeta_0^\lambda$ might be unnecessary depending on the specific application. Namely, for a pair (P, Q), we get another pair $(P, \lambda^{-1}Q)$ that is a torsion basis, with

$$e_{p+1}(P, \lambda^{-1}Q) = e_{p+1}(P, Q)^{\lambda^{-1}} = \zeta^{\lambda^{-1}} = \zeta_0.$$

As ζ_0 is a public parameter, verification requires no extra λ. However, the choice of P and Q might have been performed carefully, e.g. to make sure both have small x-coordinates to reduce communication cost. It thus depends per application if a modified torsion basis $(P, \lambda^{-1}Q)$ reduces communication cost.

Remark 4. The above algorithm not only verifies that P and Q are full-torsion points, but includes the supersingularity verification of E_A, as it shows E_A has points of order $p + 1$. This can be useful for applications, e.g. those in [7].

4.2 Pairing-based supersingularity verification

Supersingularity verification asks us to verify that E_A is a supersingular curve. A sound analysis of the performance of different algorithms was made by Banegas, Gilchrist, and Smith [2]. They examine

a. a product-tree-based approach to find a point of order $N \geq 4\sqrt{p}$,
b. Sutherland's algorithm [47] based on isogeny volcanoes, and
c. Doliskani's test [20] based on division polynomials.

They conclude that Doliskani's test is best for Montgomery models over $\mathbb{F}_p$, as it requires only a single scalar multiplication over $\mathbb{F}_{p^2}$ of length $\log p$ followed by $\log p$ squarings in $\mathbb{F}_{p^2}$. The algorithms we propose resemble the product-tree approach, but move the computation of the orders of points from the curve to the field. There are two ways to apply the pairing approach here:

Aproach 1. Sample , using Elligator [6], random points $P \in E(\mathbb{F}_p)$ and $Q \in E^t(\mathbb{F}_p)$, compute $\zeta = e_r(P,Q)$ for $r = p+1$, and compute $\mathsf{Ord}(\zeta)$ using a product-tree up until we have verified $\mathsf{Ord}(\zeta) \geq 4\sqrt{p}$.
Aproach 2. Divide L_χ into two lists L_1 and L_2 such that $\ell^{(1)} := \prod_{\ell_i \in L_1} \ell_i$ is slightly larger than $4\sqrt{p}$. Sample again two random points $P \in E(\mathbb{F}_p)$ and $Q \in E^t(\mathbb{F}_p)$, and multiply both by $\frac{p+1}{\ell^{(1)}} = h \cdot \ell^{(2)}$ so that $P, Q \in E[\ell^{(1)}]$. Then, compute the pairing of degree $r = \ell^{(1)}$ and verify that $\zeta \in \mu_r$ has $\mathsf{Ord}(\zeta) \geq 4\sqrt{p}$.

Approach 1 is essentially Algorithm 4, where we cut off the computation of $\mathsf{Ord}(\zeta)$ early whenever we have enough torsion. Approach 2 uses the fact that we do not have to work in all of μ_{p+1} to verify supersingularity. This reduces the number of steps in the Miller loop by half, $\frac{1}{2}\log p$ compared to $\log p$, but requires two Montgomery ladders of $\log(p)/2$ bits to kill the $\ell^{(2)}$-torsion of P and Q. Note that we must take L_1 a few bits larger than $4\sqrt{p}$ to ensure with high probability that random points P, Q have enough torsion to verify or falsify supersingularity. In practice, the fastest approach is highly dependent on the prime p and the number of factors ℓ_i, as well as the size of the cofactor 2^k. Approach 2 is summarized in Algorithm 5. See the folder `supersingularity` for the implementations of these algorithms.

Algorithm 5 Verification of supersingularity

```
Input: $A \in \mathbb{F}_p$, where $p = h \cdot \ell^{(1)} \cdot \ell^{(2)} - 1$
Output: True or False
1: $(P,Q) \xleftarrow{\$} E(\mathbb{F}_p) \times E^t(\mathbb{F}_p)$
2: $P \leftarrow [4 \cdot \ell^{(2)}]P$, $Q \leftarrow [h \cdot \ell^{(2)}]Q$
3: $\zeta \leftarrow \mathrm{Re}(e_{\ell^{(1)}}(P,Q))$
4: $m \leftarrow \mathsf{Order}(\zeta)$
5: if $m \geq 4\sqrt{p}$ then
6:     return True
7: else
8:     else repeat
```

Remark 5. Line 4 of Algorithm 5 computes the order of ζ using a product-tree approach to verify **(a)** $\zeta' := \zeta^{\frac{p+1}{\ell_i}} \neq 1$ and **(b)** $\zeta'^{\ell_i} = 1$. If at any moment, **(a)** holds but **(b)** does not, this implies the order of ζ does not divide $p+1$, which implies the curve is *ordinary*. In such a case, as in the original algorithm [10, Alg. 1], we return **False** as we know the curve is not supersingular.

Remark 6. Note that in an approach where torsion points (P,Q) are given, together with a discrete log λ, so that $e_{p+1}(P,Q) = \zeta_0^\lambda$, or even the variant where $(P, \lambda^{-1}Q)$ is given as in Remark 3, the cost of supersingularity verification is essentially that of a single pairing computation, or that of a pairing computation together with a Lucas exponentiation of length $\log \lambda \approx \log p$. For CSIDH-512, this beats Doliskani's test as we will see in Section 4.4

4.3 Finding full-torsion points.

Finding full-torsion points is more tricky. Current implementations [7, 13] simply sample random points and compute their order until they find a full-torsion point. Although the probability of finding a full-torsion point is not low, this approach is inefficient as the cost of computing the order per point is similar to the one sketched in Section 4.1. The curve-equivalent of Gauss's Algorithmcan be given as follows: Given a point $P \in E(\mathbb{F}_p)$ with $\mathsf{Ord}(P) \neq p+1$, sample a random point $P' \in E(\mathbb{F}_p)$. Set $P' \leftarrow [\mathsf{Ord}(P)]P'$ and compute $\mathsf{Ord}(P')$. Set $P \leftarrow P + P'$ and $\mathsf{Ord}(P) \leftarrow \mathsf{Ord}(P) \cdot \mathsf{Ord}(P')$. Repeat until P is a full-torsion point. This already improves on the naïve approach, yet still requires a lot of curve arithmetic. We apply pairings again to improve performance. We specifically need Scott-Miller's algorithm (Section 3.4) to compute a variable number of pairings for a given P.

Abstract point-of-view. From the abstract point of view, sketched in Section 3.1, we want to identify a full-torsion point P as an isomorphism $f_{rP} : E^t(\mathbb{F}_p) \to \mu_r$, using Lemma 1. Scott-Miller's algorithm allows us to compute a representation of f_{rP} and to evaluate f_{rP} efficiently on points $Q \in E^t(\mathbb{F}_p)$.

Starting from random points $P_1 \in E(\mathbb{F}_p)$ and $Q_1 \in E^t(\mathbb{F}_p)$, we compute $\zeta_1 := f_{rP_1}(P_1, Q_1)^{p-1}$ and $\mathsf{Ord}(\zeta_1)$. The missing torsion $m_1 = \mathsf{Miss}(\zeta_1)$ is then equal to $\mathrm{lcm}(\mathsf{Miss}(P), \mathsf{Miss}(Q)))$. If $m_1 = 1$, then we know both P_1 and Q_1 are full-torsion points. If $m_1 \neq 1$, we continue with a second point Q_2. Compute ζ_2 and $m_2 = \mathsf{Miss}(\zeta_2)$. Let $d = \gcd(m_1, m_2)$. If $d = 1$, that is, m_1 and m_2 are co-prime, then P_1 is a full-torsion point, and we can apply Gauss's Algorithm to compute a full-order point Q, given Q_1 and Q_2.

For $d > 1$, it is most likely that $d = |\ker f_{rP_1}| = \mathsf{Miss}(P_1)$, or, if unlucky, both Q_1 and Q_2 miss d-torsion. The probability that both Q_1 and Q_2 miss d-torsion is $\frac{1}{d^2}$. Hence, if d is small, this is unlikely but possible. If d is a large prime, we are almost certain P_1 misses d-torsion. In the former case, we sample a third point Q_3 and repeat the same procedure. In the latter case, we use Q_1 and Q_2 to compute a full-torsion Q. Using Q, we compute f_{rQ} and apply the same procedure to points P_i to create a full-torsion point P (reusing P_1).

Distinguishing between these cases is highly dependent on the value of d, which in turn depends on L_χ and p. We leave these case-dependent details to the reader. See `bigfastfinding.rs` for the implementation of this algorithm.

Remark 7. To distinguish between the cases dependent on d, it is favorable to have no small factors ℓ_i in $p+1$. This coincides with independent analysis [1, 13] that small $\ell_i \mid p+1$ might not be optimal for deterministic variants of CSIDH.

Remark 8. One can improve on randomly sampling P_1 and Q_1, as we can sample points directly in $E \setminus [2]E$ [16] by ensuring the x-coordinates are not quadratic residues in $\mathbb{F}_p$. Similar techniques from p-descent might also apply for $\ell_i > 2$.

Remark 9. The above approach is very efficient to find a second full-torsion point $Q \in E^t(\mathbb{F}_p)$ given a first full-torsion point $P \in E(\mathbb{F}_p)$, which may be of independent interest in other situations.

4.4 Concrete cost for CSIDH-512

We have implemented and evaluated the performance of the algorithms in Section 4 for p512, the prime used in CSIDH-512. Table 2 shows the performance in $\mathbb{F}_p$-operations, compared to well-known or state-of-the-art algorithms.

	Source	**M**	**S**	**A**	Total
Product-tree torsion verif.	[7]	51 318	29 388	73 396	75 562
Pairing-based torsion verif.	Alg. 4	13 693	6838	18 424	19 293
Pairing-based (given λ)	Sec. 4.1	10 472	3471	11 616	13 364
CSIDH-Supersingularity verif.	[10]	13 324	7628	19 052	19 617
Doliskani's test	[2, 20]	13 789	2	30 642	14 097
Approach 1 (pairing-based)	Sec. 4.2	11 081	4112	12 914	14 500
Approach 2 (pairing-based)	Alg. 5	9589	5801	10 434	14 334

Table 2. Concrete cost of the algorithms in this section using the prime in CSIDH-512. 'Total' gives the number of $\mathbb{F}_p$-operations, with cost model $1\mathbf{S} = 0.8\mathbf{M}$ and $1\mathbf{A} = 0.01\mathbf{M}$.

Verifying torsion points. We find that Algorithm 4 specifically for p512 takes about 19293 operations, with 12426 operations taken up by the pairing, hence order verification of ζ using Lucas exponentiation requires only 6867 operations, closely matching the predicted cost $C_{\text{Lucas}} \cdot \log n \cdot \log p$. In comparison, the currently-used method [7] to verify full-torsion points requires 75562 operations, hence we achieve a speed-up of 75%, due to the difference in cost per bit between C_{curve} and C_{Lucas}. If we include a system parameter ζ_0 and a discrete log λ, our cost drops down to 13364 operations, increasing the speedup from 75% to 82%.

Supersingularity verification. We find that Doliskani's test is still slightly faster, but our algorithms come within 2% of performance. Saving a single M or S in Dbl or Add would push Algorithm 5 below Doliskani's test for p512. When we include λ, as mentioned above, we outperform Doliskani's test by 6%.

Finding torsion points. Although the cost of this algorithm depends highly on divisors ℓ_i of $p + 1$, heuristics for p512 show that we usually only require 2 points $P_1, P_2 \in E(\mathbb{F}_p)$ and two points $Q_1, Q_2 \in E^t(\mathbb{F}_p)$, together with pairing computations $e_r(P_1, Q_1)$ and $e_R(P_1, Q_2)$ to find full-torsion points P and Q. We leave out concrete performance numbers, as this varies too much per case.

5 Applications of pairing-based algorithms

The pairing-based algorithms from Section 4 are of independent interest, but also find natural applications in (deterministic) variants of CSIDH.

Applying pairing-based algorithms. In all versions of CSIDH, supersingularity verification is required on public keys. We estimate that, depending on the shape and size of the prime p, either Doliskani's test or one of the pairing-based algorithms (Section 4.2) is optimal.

For deterministic variants of CSIDH [7, 13], including (an Elligator seed for) a torsion basis of E_A in the public key is only natural, and this is exactly what is proposed for the dCSIDH variant of [7]. This requires verification of such a torsion basis. For this verification, Algorithm 4 clearly outperforms curve-based approaches. Furthermore, the verification of such a torsion basis also verifies the supersingularity of E_A, which would otherwise have cost an additional $\mathcal{O}(\log p)$ operations, using either Doliskani's test or one of our pairing-based algorithms.

Including torsion-point information in the public key also requires a party to *find* such a torsion basis in key generation. The pairing-based approach described in Section 4.3 heuristically beats current approaches based on random sampling.

Remark 10. One might think that including a torsion basis (P, Q) resulting from the pairing-based approach in a public key would cost $2 \log p$ bits, as it requires a description of the x-coordinates of both points. However, it is possible to use points P_i and Q_i with very small x-coordinates, and to describe the construction of P (resp. Q) as a combination of the P_i (resp. Q_i) in a few bits.

Constant-time versions. Both Algorithms 4 and 5 are easy to implement in constant-time, given constant-time curve and field arithmetic. However, constant-time verification is usually not required, as both E_A and (P, Q) should be public.

For a constant-time approach to finding full-torsion points, a major roadblock is finding a constant-time version of Gauss's Algorithm. This is both mathematically interesting as well as cryptographically useful, but seems to require a better understanding of the distribution of the x-coordinates of full-torsion points for curves, or the $(p+1)$-th primitive roots for fields.

Beyond current implementations. Current deterministic variants of CSIDH [7, 11, 13] are limited to exponents $e_i \in \{-1, 0, +1\}$. Going beyond such exponents requires sampling new points during the class group action evaluation on an intermediate curve E'. To not leak any information on E' in a deterministic implementation, therefore, requires a constant-time torsion-basis algorithm as sketched above. This would allow approaches with $e_i \geq 1$ for multiple small ℓ_i to reduce the number of ℓ_i-isogenies for large ℓ_i, which is deemed favorable in constant-time probabilistic approaches [1, 14, 36].

For ordinary CSIDH and CTIDH, using full-torsion points in every round would have the further improvement that the number of rounds is constant, and we have no trial-and-error approaches in the group action computation, providing a stronger defence against certain side-channel attacks [3]. In particular for CTIDH, we can use full-torsion points to remove the "coin flip" that decides if a batch is performed. This improves performance and design properties. We leave a full analysis for future work.

A final remark should be made about the use of theta functions to compute pairings [33, 34]. Such an approach might yield speed-ups in comparison to the methods used in this work and the use of theta functions could provide a more general framework for higher dimensional isogeny-based cryptography.

References

[1] Gustavo Banegas, Daniel J. Bernstein, Fabio Campos, Tung Chou, Tanja Lange, Michael Meyer, Benjamin Smith, and Jana Sotáková. "CTIDH: faster constant-time CSIDH". In: 2021.4 (2021). https://tches.iacr.org/index.php/TCHES/article/view/9069, pp. 351–387. DOI: 10.46586/tches.v2021.i4.351-387.

[2] Gustavo Banegas, Valerie Gilchrist, and Benjamin Smith. "Efficient supersingularity testing over $\mathbb{F}_p$ and CSIDH key validation". In: *Mathematical Cryptology* 2.1 (2022), pp. 21–35.

[3] Gustavo Banegas, Juliane Krämer, Tanja Lange, Michael Meyer, Lorenz Panny, Krijn Reijnders, Jana Sotáková, and Monika Trimoska. "Disorientation faults in CSIDH". In: *Advances in Cryptology–EUROCRYPT 2023: 42nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Lyon, France, April 23-27, 2023, Proceedings, Part V*. Springer. 2023, pp. 310–342.

[4] Paulo S. L. M. Barreto, Hae Yong Kim, Ben Lynn, and Michael Scott. "Efficient Algorithms for Pairing-Based Cryptosystems". In: 2002, pp. 354–368. DOI: 10.1007/3-540-45708-9_23.

[5] Paulo S. L. M. Barreto, Ben Lynn, and Michael Scott. "Efficient Implementation of Pairing-Based Cryptosystems". In: 17.4 (Sept. 2004), pp. 321–334. DOI: 10.1007/s00145-004-0311-z.

[6] Daniel J. Bernstein, Mike Hamburg, Anna Krasnova, and Tanja Lange. "Elligator: elliptic-curve points indistinguishable from uniform random strings". In: 2013, pp. 967–980. DOI: 10.1145/2508859.2516734.

[7] Fabio Campos, Jorge Chavez-Saab, Jesús-Javier Chi-Domínguez, Michael Meyer, Krijn Reijnders, Francisco Rodríguez-Henríquez, Peter Schwabe, and Thom Wiggers. *On the Practicality of Post-Quantum TLS Using Large-Parameter CSIDH*. Cryptology ePrint Archive, Paper 2023/793. 2023. URL: https://eprint.iacr.org/2023/793.

[8] Fabio Campos, Matthias J Kannwischer, Michael Meyer, Hiroshi Onuki, and Marc Stöttinger. "Trouble at the CSIDH: protecting CSIDH with dummy-operations against fault injection attacks". In: *2020 Workshop on Fault Detection and Tolerance in Cryptography (FDTC)*. IEEE. 2020, pp. 57–65.

[9] Fabio Campos, Michael Meyer, Krijn Reijnders, and Marc Stöttinger. *Patient Zero and Patient Six: Zero-Value and Correlation Attacks on CSIDH and SIKE*. IACR Cryptology ePrint Archive, Report 2022/904. To appear in SAC 2022. 2022. URL: https://eprint.iacr.org/2022/904.

[10] Wouter Castryck, Tanja Lange, Chloe Martindale, Lorenz Panny, and Joost Renes. "CSIDH: An Efficient Post-Quantum Commutative Group Action". In: 2018, pp. 395–427. DOI: 10.1007/978-3-030-03332-3_15.

[11] Daniel Cervantes-Vázquez, Mathilde Chenu, Jesús-Javier Chi-Domínguez, Luca De Feo, Francisco Rodríguez-Henríquez, and Benjamin Smith. "Stronger and faster side-channel protections for CSIDH". In: *Progress in Cryptology–LATINCRYPT 2019: 6th International Conference on Cryp-*

tology and Information Security in Latin America, Santiago de Chile, Chile, October 2–4, 2019, Proceedings 6. Springer. 2019, pp. 173–193.

[12] Sanjit Chatterjee, Palash Sarkar, and Rana Barua. "Efficient Computation of Tate Pairing in Projective Coordinate over General Characteristic Fields". In: 2005, pp. 168–181.

[13] Jorge Chávez-Saab, Jesús-Javier Chi-Domínguez, Samuel Jaques, and Francisco Rodríguez-Henríquez. "The SQALE of CSIDH: sublinear Vélu quantum-resistant isogeny action with low exponents". In: *Journal of Cryptographic Engineering* 12.3 (2022), pp. 349–368.

[14] Jesús-Javier Chi-Domínguez and Francisco Rodríguez-Henríquez. "Optimal strategies for CSIDH". In: *Adv. Math. Commun.* 16.2 (2022), pp. 383–411. DOI: 10.3934/amc.2020116. URL: https://doi.org/10.3934/amc.2020116.

[15] Craig Costello. *Pairings for beginners*. 2015. URL: https://www.craigcostello.com.au/.

[16] Craig Costello, David Jao, Patrick Longa, Michael Naehrig, Joost Renes, and David Urbanik. "Efficient compression of SIDH public keys". In: *Advances in Cryptology–EUROCRYPT 2017: 36th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Paris, France, April 30–May 4, 2017, Proceedings, Part I 36*. Springer. 2017, pp. 679–706.

[17] Craig Costello, Patrick Longa, and Michael Naehrig. "Efficient Algorithms for Supersingular Isogeny Diffie-Hellman". In: 2016, pp. 572–601. DOI: 10.1007/978-3-662-53018-4_21.

[18] Craig Costello and Benjamin Smith. "Montgomery curves and their arithmetic - The case of large characteristic fields". In: 8.3 (Sept. 2018), pp. 227–240. DOI: 10.1007/s13389-017-0157-6.

[19] Luca De Feo, David Kohel, Antonin Leroux, Christophe Petit, and Benjamin Wesolowski. "SQISign: Compact Post-quantum Signatures from Quaternions and Isogenies". In: 2020, pp. 64–93. DOI: 10.1007/978-3-030-64837-4_3.

[20] Javad Doliskani. "On division polynomial PIT and supersingularity". In: *Applicable Algebra in Engineering, Communication and Computing* 29.5 (2018), pp. 393–407.

[21] SD Galbraith. *Pairings. London Mathematics Society Lecture Note Series, vol. 317*. 2005.

[22] Steven D Galbraith and Xibin Lin. "Computing pairings using x-coordinates only". In: *Designs, Codes and Cryptography* 50.3 (2009), pp. 305–324.

[23] Aurore Guillevic. "Comparing the Pairing Efficiency over Composite-Order and Prime-Order Elliptic Curves". In: 2013, pp. 357–372. DOI: 10.1007/978-3-642-38980-1_22.

[24] Ryuichi Harasawa, Junji Shikata, Joe Suzuki, and Hideki Imai. "Comparing the MOV and FR Reductions in Elliptic Curve Cryptography". In: 1999, pp. 190–205. DOI: 10.1007/3-540-48910-X_14.

[25] Aaron Hutchinson, Jason LeGrow, Brian Koziel, and Reza Azarderakhsh. "Further optimizations of CSIDH: a systematic approach to efficient strategies, permutations, and bound vectors". In: *Applied Cryptography and Network Security: 18th International Conference, ACNS 2020, Rome, Italy, October 19–22, 2020, Proceedings, Part I 18*. Springer. 2020, pp. 481–501.

[26] Antoine Joux. "A One Round Protocol for Tripartite Diffie-Hellman". In: 17.4 (Sept. 2004), pp. 263–276. DOI: 10.1007/s00145-004-0312-y.

[27] Marc Joye and Jean-Jacques Quisquater. "On the Importance of Securing Your Bins: The Garbage-man-in-the-middle Attack". In: 1997, pp. 135–141. DOI: 10.1145/266420.266449.

[28] Marc Joye and Sung-Ming Yen. "The Montgomery Powering Ladder". In: 2003, pp. 291–302. DOI: 10.1007/3-540-36400-5_22.

[29] Yutaro Kiyomura and Tsuyoshi Takagi. "Efficient algorithm for Tate pairing of composite order". In: *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences* 97.10 (2014), pp. 2055–2063.

[30] Tetsutaro Kobayashi, Kazumaro Aoki, and Hideki Imai. "Efficient algorithms for Tate pairing". In: *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences* 89.1 (2006), pp. 134–143.

[31] Jason T LeGrow and Aaron Hutchinson. "(Short Paper) Analysis of a Strong Fault Attack on Static/Ephemeral CSIDH". In: *Advances in Information and Computer Security: 16th International Workshop on Security, IWSEC 2021, Virtual Event, September 8–10, 2021, Proceedings*. Springer. 2021, pp. 216–226.

[32] Kaizhan Lin, Weize Wang, Zheng Xu, and Chang-An Zhao. *A Faster Software Implementation of SQISign*. Cryptology ePrint Archive, Paper 2023/753. https://eprint.iacr.org/2023/753. 2023. URL: https://eprint.iacr.org/2023/753.

[33] David Lubicz and Damien Robert. "A generalisation of Miller's algorithm and applications to pairing computations on abelian varieties". In: *Journal of Symbolic Computation* 67 (2015), pp. 68–92.

[34] David Lubicz and Damien Robert. "Efficient pairing computation with theta functions". In: *International Algorithmic Number Theory Symposium*. Springer. 2010, pp. 251–269.

[35] Robert J McEliece. *Finite fields for computer scientists and engineers*. Vol. 23. Springer Science & Business Media, 2012.

[36] Michael Meyer, Fabio Campos, and Steffen Reith. "On lions and elligators: An efficient constant-time implementation of CSIDH". In: *Post-Quantum Cryptography: 10th International Conference, PQCrypto 2019, Chongqing, China, May 8–10, 2019 Revised Selected Papers 10*. Springer. 2019, pp. 307–325.

[37] Michael Meyer and Steffen Reith. "A faster way to the CSIDH". In: *Progress in Cryptology–INDOCRYPT 2018: 19th International Conference on Cryptology in India, New Delhi, India, December 9–12, 2018, Proceedings 19*. Springer. 2018, pp. 137–152.

[38] Victor S. Miller. “The Weil Pairing, and Its Efficient Calculation”. In: 17.4 (Sept. 2004), pp. 235–261. DOI: 10.1007/s00145-004-0315-8.

[39] Hiroshi Onuki, Yusuke Aikawa, Tsutomu Yamazaki, and Tsuyoshi Takagi. “(Short Paper) A Faster Constant-Time Algorithm of CSIDH Keeping Two Points”. In: 2019, pp. 23–33. DOI: 10.1007/978-3-030-26834-3_2.

[40] George W Reitwiesner. “Binary arithmetic”. In: *Advances in computers.* Vol. 1. Elsevier, 1960, pp. 231–308.

[41] Michael Scott. “Computing the Tate Pairing”. In: 2005, pp. 293–304. DOI: 10.1007/978-3-540-30574-3_20.

[42] Michael Scott. “Pairing implementation revisited”. In: *Cryptology ePrint Archive* (2019).

[43] Michael Scott. *Understanding the Tate pairing.* 2004. URL: http://www.computing.dcu.ie/~mike/tate.html.

[44] Michael Scott and Paulo S. L. M. Barreto. “Compressed Pairings”. In: 2004, pp. 140–156. DOI: 10.1007/978-3-540-28628-8_9.

[45] Joseph H Silverman. “A survey of local and global pairings on elliptic curves and abelian varieties”. In: *Pairing-Based Cryptography-Pairing 2010: 4th International Conference, Yamanaka Hot Spring, Japan, December 2010. Proceedings 4.* Springer. 2010, pp. 377–396.

[46] Katherine E Stange. “The Tate pairing via elliptic nets”. In: *Pairing Based Cryptography.* Springer. 2007, pp. 329–348.

[47] Andrew V Sutherland. “Identifying supersingular elliptic curves”. In: *LMS Journal of Computation and Mathematics* 15 (2012), pp. 317–325.

[48] Jacques Vélu. “Isogénies entre courbes elliptiques”. In: *Comptes Rendus de l’Académie des Sciences de Paris, Séries A* 273 (1971), pp. 238–241.

[49] Frederik Vercauteren. “Optimal pairings”. In: *IEEE transactions on information theory* 56.1 (2009), pp. 455–461.

A Subalgorithms of Miller's algorithm

The following algorithms give an algorithmic description of the subroutines Dbl, Add and Sub as used in the optimizations of the Miller loop.

Algorithm 6 Subroutine Dbl in the Miller loop

```
Input: $T \in \mathbb{F}_p^4, f \in \mathbb{F}_p^2, x_Q, y_Q \in \mathbb{F}_p, A \in \mathbb{F}_p$
Output: $(T, f)$ corresponding to the doubling $T \leftarrow T + T,\ f \leftarrow f^2 \cdot \ell_{T,T}(Q)$
1: $(T, \ell) \leftarrow$ DblAndLine$(T, A)$                // Doubles $T$ and computes $\ell_{T,T}$
2: $(\alpha, \beta) \leftarrow$ Eval$(\ell, x_Q, y_Q)$           // Evaluates $\ell_{T,T}(Q)$
3: $f \leftarrow$ FP2SQR$(f)$
4: $f \leftarrow$ FP2MUL$(f, (\alpha, \beta))$
5: return $T, f$
```

Algorithm 7 Subroutine Add in the Miller loop

```
Input: $T \in \mathbb{F}_p^4, f \in \mathbb{F}_p^2, x_P, y_P, x_Q, y_Q \in \mathbb{F}_p, A \in \mathbb{F}_p$
Output: $(T, f)$ corresponding to the addition $T \leftarrow T + P,\ f \leftarrow f \cdot \ell_{T,P}(Q)$
1: $(T, \ell) \leftarrow$ AddAndLine$(T, x_P, y_P, A)$          // Adds $T + P$ and computes $\ell_{T,P}$
2: $(\alpha, \beta) \leftarrow$ Eval$(\ell, x_Q, y_Q)$           // Evaluates $\ell_{T,P}(Q)$
3: $f \leftarrow$ FP2MUL$(f, (\alpha, \beta))$
4: return $T, f$
```

Algorithm 8 Subroutine Sub in the Miller loop

```
Input: $T \in \mathbb{F}_p^4, f \in \mathbb{F}_p^2, x_P, y_P, x_Q, y_Q \in \mathbb{F}_p, A \in \mathbb{F}_p$
Output: $(T, f)$ corresponding to the substraction $T \leftarrow T - P,\ f \leftarrow f \cdot \ell_{T,-P}(Q)$
1: $(T, f) \leftarrow$ Add$(T, f, x_P, y_P, x_Q, y_Q, A)$
2: return $T, f$
```

B Subalgorithms of Scott-Miller's algorithm

We describe here Scott-Miller's subalgorithms Construct, Evaluate and Exponentiate.

Algorithm 9 Scott-Miller's subalgorithm Construct

```
Input: $x_P, y_p \in \mathbb{F}_p$, $p + 1 = \sum_{i=0}^{t} t_i \cdot 2^i$
Output: A representation of $f_{rP}$ as an array of line functions
 1: $T = (X^2, XZ, Z^2, YZ) \leftarrow (x_P^2, x_P, 1, y_P)$
 2: $f \leftarrow [\ ]$
 3: for $i$ from $t - 1$ to 0 do
 4:     $T, l \leftarrow$ Dbl$(T)$
 5:     Append $l$ to $f$
 6:     if $t_i = 1$ then
 7:         $T, l \leftarrow$ Add$(T, f, x_P, y_P)$
 8:         Append $l$ to $f$
 9:     if $t_i = -1$ then
10:         $T, l \leftarrow$ Sub$(T, f, x_P, y_P)$
11:         Append $l$ to $f$
12: return $f$
```

Algorithm 10 Scott-Miller's subalgorithm Evaluate

```
Input: A representation of $f_{rP}$ as an array of line functions, and a point $x_Q, y_Q \in \mathbb{F}_p$
Output: The unreduced Tate evaluation $f_{rP}(Q) \in \mathbb{F}_{p^2}$
 1: $f_0 \leftarrow (1, 0)$
 2: for $l$ in $f$ do
 3:     if $l$ is a doubling then $f_0 \leftarrow f_0^2$
 4:     $f_0 \leftarrow f_0 \cdot$ Eval$(l, x_Q, y_Q)$
 5: return $f_0$
```

Algorithm 11 Scott-Miller's subalgorithm Exponentiate

```
Input: The unreduced Tate pairing $f_0 = a + bi$ as a pair $f = (a, b)$
Output: The reduced Tate pairing $\zeta \in \mu_r$
1: $a \leftarrow f[0]$, $b \leftarrow f[1]$
2: $\zeta \leftarrow \frac{a^2-b^2}{a^2+b^2}$
3: return $\zeta$
```

The composition of these three algorithms is referred to as Scott-Miller's algorithm.