Stronger and Faster Side-Channel Protections for CSIDH

Daniel Cervantes-Vázquez[1], Mathilde Chenu[2,3], Jesús-Javier Chi-Domínguez[1], Luca De Feo[4], Francisco Rodríguez-Henríquez[1], Benjamin Smith[3,2]

[1] CINVESTAV - Centro de Investigaciòn y de Estudios Avanzados del Instituto Politécnico Nacional, Mexico City, Mexico
[2] École polytechnique, Institut Polytechnique de Paris, Palaiseau, France
[3] Inria, équipe-projet GRACE, Université Paris–Saclay, France
[4] Université Paris Saclay – UVSQ, Versailles, France

Abstract. CSIDH is a recent quantum-resistant primitive based on the difficulty of finding isogeny paths between supersingular curves. Recently, two constant-time versions of CSIDH have been proposed: first by Meyer, Campos and Reith, and then by Onuki, Aikawa, Yamazaki and Takagi. While both offer protection against timing attacks and simple power consumption analysis, they are vulnerable to more powerful attacks such as fault injections. In this work, we identify and repair two oversights in these algorithms that compromised their constant-time character. By exploiting Edwards arithmetic and optimal addition chains, we produce the fastest constant-time version of CSIDH to date. We then consider the stronger attack scenario of fault injection, which is relevant for the security of CSIDH static keys in embedded hardware. We propose and evaluate a dummy-free CSIDH algorithm. While these CSIDH variants are slower, their performance is still within a small constant factor of less-protected variants. Finally, we discuss derandomized CSIDH algorithms.

1 Introduction

Isogeny-based cryptography was introduced by Couveignes [10], who defined a key exchange protocol similar to Diffie–Hellman based on the action of an ideal class group on a set of ordinary elliptic curves. Couveignes' protocol was independently rediscovered by Rostovtsev and Stolbunov [27,28], who were the first to recognize its potential as a post-quantum candidate. Recent efforts to make this system practical have put it back at the forefront of research in post-quantum cryptography [13]. A major breakthrough was achieved by Castryck, Lange, Martindale, Panny, and Renes with CSIDH [6], a reinterpretation of Couveignes' system using supersingular curves defined over a prime field.

The first implementation of CSIDH completed a key exchange in less than 0.1 seconds, and its performance has been further improved by Meyer and Reith [22]. However, both [6] and [22] recognized the difficulty of implementing CSIDH with constant-time algorithms, that is, algorithms whose running time, sequence of operations, and memory access patterns do not depend on secret data. The implementations of [6] and [22] are thus vulnerable to simple timing attacks.

The first attempt at implementing CSIDH in constant-time was realized by Bernstein, Lange, Martindale, and Panny [3], but their goal was to obtain a fully deterministic reversible circuit implementing the class group action, to be used in quantum cryptanalyses. The distinct problem of efficient CSIDH implementation with side-channel protection was first tackled by Jalali, Azarderakhsh, Mozaffari Kermani, and Jao [16], and independently by Meyer, Campos, and Reith [21], whose work was improved by Onuki, Aikawa, Yamazaki, and Takagi [26].

The approach of Jalali *et al.* is similar to that of [3], in that they achieve a stronger notion of constant time (running time independent from *all* inputs), at the cost of allowing the algorithm to fail with a small probability. In order to make the failure probability sufficiently low, they introduce a large number of useless operations, which make the performance significantly worse than the original CSIDH algorithm. This poor performance and possibility of failure reduces the interest of this implementation; we will not analyze it further here.

Meyer *et al.* take a different path: the running time of their algorithm is independent of the secret key, but not of the output of an internal random number generator. They claim a speed only 3.10 times slower than the unprotected algorithm in [22]. Onuki *et al.* introduced new improvements, claiming a speed-up of 27.35% over Meyer *et al.*, i.e., a net slow-down factor of 2.25 compared to [22].

Our contribution. In this work we take a new look at side-channel protected implementations of CSIDH. We start by reviewing the implementations in [21] and [26]. We highlight some flaws that make their constant-time claims disputable, and propose fixes for them. Since these fixes introduce some minor slow-downs, we report on the performance of the revised algorithms.

Then, we introduce new optimizations to make both [21] and [26] faster: we improve isogeny formulas for the model, and we introduce the use of optimal addition chains in the scalar multiplications. With these improvements, we obtain a version of CSIDH protected against timing and some simple power analysis (SPA) attacks that is 25% more efficient than [21] and 15% more efficient than a repaired version of [26].

Then, we shift our focus to stronger security models. All constant-time versions of CSIDH presented so far use so-called "dummy operations", i.e., computations whose result is not used, but whose role is to hide the conditional structure of the algorithm from timing and SPA attacks that read the sequence of operations performed from a single power trace. However, this countermeasure is easily defeated by fault-injection attacks, where the adversary may modify values during the computation. We propose a new constant-time variant of CSIDH without dummy operations as a first-line defence. The new version is only twice as slow as the simple constant-time version.

We conclude with a discussion of derandomized variants of CSIDH. The versions discussed previously are "constant-time" in the sense that their running time is uncorrelated to the secret key, however it depends on some (necessarily secret) seed to a PRNG. While this notion of "constant-time" is usually considered good enough for side-channel protection, one may object that a compromise of the PRNG or the seed generation would put the security of the implemen-

tation at risk, even if the secret was securely generated beforehand (with an uncomprised PRNG) as part of a long-term or static keypair. We observe that this dependence on additional randomness is not necessary: a simple modification of CSIDH, already considered in isogeny-based signature schemes [11,14], can easily be made constant-time and free of randomness. Unfortunately this modification requires increasing substantially the size of the base field, and is thus considerably slower and not compatible with the original version. On the positive side, the increased field size makes it much more resistant to quantum attacks, a non-negligible asset in a context where the quantum security of CSIDH is still unclear; it can thus be seen as CSIDH variant for the paranoid.

Organization. In §2 we briefly recall ideas, algorithms and parameters from CSIDH [6]. In §3 we highlight shortcomings in [21] and [26] and propose ways to fix them. In §4 we introduce new optimizations compatible with all previous versions of CSIDH. In §5 we introduce a new algorithm for evaluating the CSIDH group action that is resistant against timing and some simple power analysis attacks, while providing protection against some fault injections. Finally, in §6 we discuss a more costly variant of CSIDH with stronger security guarantees.

Notation. **M**, **S**, and **A** denote the cost of computing a single multiplication, squaring, and addition (or subtraction) in $\mathbb{F}_p$, respectively. We assume that a constant-time equality test $\texttt{isequal}(X, Y)$ is defined, returning 1 if $X = Y$ and 0 otherwise. We also assume that a constant-time conditional swap $\texttt{cswap}(X, Y, b)$ is defined, exchanging (X, Y) if $b = 1$ (and not if $b = 0$).

2 CSIDH

CSIDH is an isogeny based primitive, similar to Diffie–Hellman, that can be used for key exchange and encapsulation [6], signatures [11,14,4], and other more advanced protocols. Compared to the other main isogeny-based primitive SIDH [17,12], CSIDH is slower. On the positive side, CSIDH has smaller public keys, is based on a better understood security assumption, and supports an easy key validation procedure, making it better suited than SIDH for CCA-secure encryption, static-dynamic and static-static key exchange. In this work we will use the jargon of key exchange when we refer to cryptographic concepts.

CSIDH works over a finite field $\mathbb{F}_p$, where p is a prime of the special form

$$p := 4 \prod_{i=1}^{n} \ell_i - 1$$

with $\ell_1, \ldots, \ell_n$ a set of small odd primes. Concretely, the original CSIDH article [6] defined a 511-bit p with $\ell_1, \ldots, \ell_{n-1}$ the first 73 odd primes, and $\ell_n = 587$.

The set of public keys in CSIDH is a subset of all supersingular elliptic curves defined over $\mathbb{F}_p$, in *Montgomery form* $y^2 = x^3 + Ax^2 + x$, where $A \in \mathbb{F}_p$ is

called the *A-coefficient* of the curve.[5] The endomorphism rings of these curves are isomorphic to orders in the imaginary quadratic field $\mathbb{Q}(\sqrt{-4p})$. Castryck *et al.* [6] choose to restrict the public keys to the *horizontal isogeny class* of the curve with $A = 0$, so that all endomorphism rings are isomorphic to $\mathbb{Z}[\sqrt{-p}]$.

2.1 The class group action

Let $E/\mathbb{F}_p$ be an elliptic curve with $\mathrm{End}(E) \cong \mathbb{Z}[\sqrt{-p}]$. If $\mathfrak{a}$ is a nonzero ideal in $\mathbb{Z}[\sqrt{-p}]$, then it defines a finite subgroup $E[\mathfrak{a}] = \bigcap_{\alpha\in\mathfrak{a}} \ker(\alpha)$, where we identify each α with its image in $\mathrm{End}(E)$. We then have a quotient isogeny $\phi : E \to E' = E/E[\mathfrak{a}]$ with kernel $\mathfrak{a}$; this isogeny and its codomain is well-defined up to isomorphism. If $\mathfrak{a} = (\alpha)$ is principal, then $\phi \cong \alpha$ and $E/E[\mathfrak{a}] \cong E$. Hence, we get an action of the ideal class group $\mathrm{Cl}(\mathbb{Z}[\sqrt{-p}])$ on the set of isomorphism classes of elliptic curves E over $\mathbb{F}_p$ with $\mathrm{End}(E) \cong \mathbb{Z}[\sqrt{-p}]$; this action is faithful and transitive. We write $\mathfrak{a} * E$ for the image of (the class of) E under the action of $\mathfrak{a}$, which is (the class of) $E/E[\mathfrak{a}]$ above.

For CSIDH, we are interested in computing the action of small prime ideals. Consider one of the primes ℓ_i dividing $p+1$; the principal ideal $(\ell_i) \subset \mathbb{Z}[\sqrt{-p}]$ splits into two primes, namely $\mathfrak{l}_i = (\ell_i, \pi - 1)$ and $\bar{\mathfrak{l}}_i = (\ell_i, \pi + 1)$, where π is the element of $\mathbb{Z}[\sqrt{-p}]$ mapping to the Frobenius endomorphism of the curves. Since $\bar{\mathfrak{l}}_i\mathfrak{l}_i = (\ell_i)$ is principal, we have $\bar{\mathfrak{l}}_i = \mathfrak{l}_i^{-1}$ in $\mathrm{Cl}(\mathbb{Z}[\sqrt{-p}])$, and hence

$$\bar{\mathfrak{l}}_i * (\mathfrak{l}_i * E) = \mathfrak{l}_i * (\bar{\mathfrak{l}}_i * E) = E$$

for all $E/\mathbb{F}_p$ with $\mathrm{End}(E) \cong \mathbb{Z}[\sqrt{-p}]$.

2.2 The CSIDH algorithm

At the heart of CSIDH is an algorithm that evaluates the class group action described above on any supersingular curve over $\mathbb{F}_p$. Cryptographically, this plays the same role as modular exponentiation in classic Diffie–Hellman.

The input to the algorithm is an elliptic curve $E : y^2 = x^3 + Ax^2 + x$, represented by its A-coefficient, and an ideal class $\mathfrak{a} = \prod_{i=1}^{n} \mathfrak{l}_i^{e_i}$, represented by its list of exponents $(e_i, \ldots, e_n) \in \mathbb{Z}^n$. The output is the ($A$-coefficient of the) elliptic curve $\mathfrak{a} * E = \mathfrak{l}_1^{e_1} * \cdots * \mathfrak{l}_n^{e_n} * E$.

The isogenies corresponding to $\mathfrak{l}_i = (\ell_i, \pi - 1)$ can be efficiently computed using Vélu's formulæ and their generalizations: exploiting the fact that $\#E(\mathbb{F}_p) = p + 1 = 4\prod \ell_i$, one looks for a point R of order ℓ_i in $E(\mathbb{F}_p)$ (i.e., a point that is in the kernels of both the multiplication-by-ℓ_i map and $\pi - 1$), computes the isogeny $\phi : E \to E/\langle R\rangle$ with kernel $\langle R\rangle$, and sets $\mathfrak{l}_i * E = E/\langle R\rangle$. Iterating this procedure lets us compute $\mathfrak{l}_i^e * E$ for any exponent $e \geq 0$.

The isogenies corresponding to $\mathfrak{l}_i^{-1}$ are computed in a similar fashion: this time one looks for a point R of order ℓ_i in the kernel of $\pi+1$, i.e., a point in $E(\mathbb{F}_{p^2})$ of the form (x, iy) where both x and y are in $\mathbb{F}_p$ (since $i = \sqrt{-1}$ is in $\mathbb{F}_{p^2} \setminus \mathbb{F}_p$ and satisfies $i^p = -i$). Then one proceeds as before, setting $\mathfrak{l}_i^{-1} * E = E/\langle R\rangle$.

[5] Following [8], we represent $A = A'/C'$ as a projective point $(A' : C')$; see §4.1.1.

In the sequel we assume that we are given an algorithm `QuotientIsogeny` which, given a curve $E/\mathbb{F}_p$ $\phi : E \to E' \cong E/\langle R \rangle$, and returns the pair (ϕ, E'). We refer to this operation as *isogeny computation*. Algorithm 1, taken from the original CSIDH article [6], computes the class group action.

Algorithm 1: The original CSIDH class group action algorithm for supersingular curves over $\mathbb{F}_p$ where $p = 4\prod_{i=1}^{n} \ell_i - 1$. The choice of ideals $\mathfrak{l}_i = (\ell_i, \pi - 1)$, where π is the element of $\mathbb{Q}(\sqrt{-p})$ is mapped to the p-th power Frobenius endomorphism on each curve in the isogeny class, is a system parameter. This algorithm constructs exactly $|e_i|$ isogenies for each ideal $\mathfrak{l}_i$.

Input: $A \in \mathbb{F}_p$ such that $E_A : y^2 = x^3 + Ax^2 + x$ is supersingular, and an integer exponent vector $(e_1, \ldots, e_n)$
Output: B such that $E_B : y^2 = x^3 + Bx^2 + x$ is $\mathfrak{l}_1^{e_1} * \cdots * \mathfrak{l}_n^{e_n} * E_A$,

$B \leftarrow A$
while *some* $e_i \neq 0$ **do**
  Sample a random $x \in \mathbb{F}_p$
  $s \leftarrow +1$ if $x^3 + Bx^2 + x$ is square in $\mathbb{F}_p$, else $s \leftarrow -1$
  $S \leftarrow \{i \mid e_i \neq 0, \mathrm{sign}(e_i) = s\}$
  if $S \neq \emptyset$ **then**
    $k \leftarrow \prod_{i \in S} \ell_i$
    $Q \leftarrow [(p+1)/k]P$, where P is the projective point with x-coordinate x.
    for $i \in S$ **do**
      $R \leftarrow [k/\ell_i]Q$ `// Point to be used as kernel generator`
      if $R \neq \infty$ **then**
        $(E_B, \phi) \leftarrow$ `QuotientIsogeny`(E_B, R)
        $Q \leftarrow \phi(Q)$
        $(k, e_i) \leftarrow (k/\ell_i, e_i - s)$
return B

For cryptographic purposes, the exponent vectors $(e_1, \ldots, e_n)$ must be taken from a space of size at least $2^{2\lambda}$, where λ is the (classical) security parameter. The CSIDH-512 parameters in [6] take $n = 74$, and all e_i in the interval $[-5, 5]$, so that $74 \log_2(2 \cdot 5 + 1) \simeq 255.99$, consistent with the NIST-1 security level. With this choice, the implementation of [6] computes one class group action in 40 ms on average. Meyer and Reith [22] further improved this to 36 ms on average. Neither implementation is constant-time.

2.3 The Meyer–Campos–Reith constant-time algorithm

As Meyer, Campos and Reith observe in [21], Algorithm 1 performs fewer scalar multiplications when the key has the same number of positive and negative exponents than it does in the unbalanced case where these numbers differ. Algorithm 1 thus leaks information about the distribution of positive and negative

exponents under timing attacks. Besides this, analysis of power traces would reveal the cost of each isogeny computation, and the number of such isogenies computed, which would leak the exact exponents of the private key.

In view of this vulnerability, Meyer, Campos and Reith proposed in [21] a constant-time CSIDH algorithm whose running time does not depend on the private key (though, unlike [16], it still varies due to randomness). The essential differences between the algorithm of [21] and classic CSIDH are as follows. First, to address the vulnerability to timing attacks, they choose to use only positive exponents in $[0, 10]$ for each ℓ_i, instead of $[-5, 5]$ in the original version, while keeping the same prime $p = \prod_{i=1}^{74} \ell_i - 1$. To mitigate power consumption analysis attacks, their algorithm always computes the maximal amount of isogenies allowed by the exponent, using dummy isogeny computations if needed.

Since these modifications generally produce more costly group action computations, the authors also provide several optimizations that limit the slow-down in their algorithm to a factor of 3.10 compared to [22]. These include the Elligator 2 map of [2] and [3], multiple batches for isogeny computation (SIMBA), and sample the exponents e_i from intervals of different sizes depending on ℓ_i.

2.4 The Onuki–Aikawa–Yamazaki–Takagi constant-time algorithm

Still assuming that the attacker can perform only power consumption analysis and timing attacks, Onuki, Aikawa, Yamazaki and Takagi proposed a faster constant-time version of CSIDH in [26].

The key idea is to use two points to evaluate the action of an ideal, one in $\ker(\pi - 1)$ (i.e., in $E(\mathbb{F}_p)$) and one in $\ker(\pi + 1)$ (i.e., in $E(\mathbb{F}_{p^2})$ with x-coordinate in $\mathbb{F}_p$). This allows them to avoid timing attacks, while keeping the same primes and exponent range $[-5, 5]$ as in the original CSIDH algorithm. Their algorithm also employs dummy isogenies to mitigate some power analysis attacks, as in [21]. With these improvements, they achieve a speed-up of 27.35% compared to [21].

We include pseudo-code for the algorithm of [26] in Algorithm 2, to serve both as a reference for a discussion of some subtle leaks in §3 and also as a departure point for our dummy-free algorithm in §5.

3 Repairing constant-time versions

3.1 Projective Elligator

Both [21] and [26] use the Elligator 2 map to sample a random point on the current curve E_A in step 6 of Algorithm 2. Elligator takes as input a random field element $u \in \{2, \ldots, \frac{p-1}{2}\}$ and the Montgomery A-coefficient from the current curve and returns a pair of points in $E_A[\pi - 1]$ and $E_A[\pi + 1]$ respectively.

To avoid a costly inversion of $u^2 - 1$, instead of sampling u randomly, Meyer, Campos and Reith[6] follow [3] and precompute a set of ten pairs $(u, (u^2 - 1)^{-1})$;

[6] Presumably, Onuki *et al.* do the same, however their exposition is not clear on this point, and we do not have access to their code.

Algorithm 2: The Onuki–Aikawa–Yamazaki–Takagi CSIDH algorithm for supersingular curves over $\mathbb{F}_p$, where $p = 4\prod_{i=1}^{n} \ell_i - 1$. The ideals $\mathfrak{l}_i = (\ell_i, \pi - 1)$, where π maps to the p-th power Frobenius endomorphism on each curve, and the exponent bound vector $(m_1, \ldots, m_n)$, are system parameters. This algorithm computes exactly m_i isogenies for each ℓ_i.

Input: A supersingular curve $E_A\colon y^2 = x^3 + Ax^2 + x$ over $\mathbb{F}_p$, and an integer exponent vector $(e_1, \ldots, e_n)$ with each $e_i \in [-m_i, m_i]$.
Output: $E_B\colon y^2 = x^3 + Bx^2 + x$ such that $E_B = \mathfrak{l}_1^{e_1} * \cdots * \mathfrak{l}_n^{e_n} * E_A$.

1 $(e'_1, \ldots, e'_n) \leftarrow (m_i - |e_1|, \ldots, m_i - |e_n|)$ `// Number of dummy computations`
2 $E_B \leftarrow E_A$
3 **while** *some* $e_i \neq 0$ *or* $e'_i \neq 0$ **do**
4 $S \leftarrow \{i \mid e_i \neq 0 \text{ or } e'_i \neq 0\}$
5 $k \leftarrow \prod_{i \in S} \ell_i$
6 $(T_-, T_+) \leftarrow$ `Elligator`(E_B, u) `//` $T_- \in E_B[\pi - 1]$ `and` $T_+ \in E_B[\pi + 1]$
7 $(P_0, P_1) \leftarrow \big([(p+1)/k]T_+, [(p+1)/k]T_-\big)$
8 **for** $i \in S$ **do**
9 $s \leftarrow \operatorname{sign}(e_i)$ `// Ideal` $\mathfrak{l}_i^s$ `to be used`
10 $Q \leftarrow [k/\ell_i]P_{\frac{1-s}{2}}$ `// Secret kernel point generator`
11 $P_{\frac{1+s}{2}} \leftarrow [\ell_i]P_{\frac{1+s}{2}}$ `// Secret point to be multiplied`
12 **if** $Q \neq \infty$ **then**
13 **if** $e_i \neq 0$ **then**
14 $(E_B, \varphi) \leftarrow$ `QuotientIsogeny`(E_B, Q)
15 $(P_0, P_1) \leftarrow \big(\varphi(P_0), \varphi(P_1)\big)$
16 $e_i \leftarrow e_i - s$.
17 **else**
18 $E_B \leftarrow E_B$; $P_{\frac{1-s}{2}} \leftarrow [\ell_i]P_{\frac{1-s}{2}}$; $e'_i \leftarrow e'_i - 1$ `// Dummies`
19 $k \leftarrow k/\ell_i$
20 **return** B

they try them in order until one that produces a point Q passing the test in Step 12 is found. When this happens, the algorithm moves to the next curve, and Elligator can keep on using the next precomputed value of u, going back to the first value when the tenth has been reached. This is a major departure from [3], where *all* precomputed values of u are tried *for each isogeny computation*, and the algorithm succeeds if at least one passes the test. And indeed the implementation of [21] leaks information on the secret via the timing channel:[7] since Elligator uses no randomness for u, its output only depends on the A-coefficient of the current curve, which itself depends on the secret key; but the running time of the algorithm varies and, not being correlated to u, it is necessarily correlated to A and thus to the secret.

[7] The Elligator optimization is described in §5.3 of [21]. The unoptimized constant-time version described in Algorithm 2 therein is not affected by this problem.

Fortunately this can be easily fixed by (re)introducing randomness in the input to Elligator. To avoid field inversions, we use a projective variant: given $u \neq 0, 1$ and assuming $A \neq 0$, we write $V = (A : u^2 - 1)$, and we want to determine whether V is the abscissa of a projective point on E_A. Plugging V into the homogeneous equation

$$E_A : Y^2Z^2 = X^3Z + AX^2Z^2 + XZ^3$$

gives

$$Y^2(u^2 - 1)^2 = \big((A^2u^2 + (u^2 - 1)^2\big)A(u^2 - 1).$$

We can test the existence of a solution for Y by computing the Legendre symbol of the right hand side: if it is a square, the points with projective XZ-coordinates

$$T_+ = (A : u^2 - 1), \qquad T_- = (-Au^2 : u^2 - 1)$$

are in $E_A[\pi - 1]$ and $E_A[\pi + 1]$ respectively, otherwise their roles are swapped.

We are left with the case $A = 0$. Following [3], Meyer, Campos and Reith precompute once and for all a pair of generators T_+, T_- of $E_0[\pi - 1]$ and $E_0[\pi + 1]$, and output those instead of random points. This choice suffers from a similar issue to the previous one: because the points are output in a deterministic way, the running time of the whole algorithm will be correlated to the number of times the curve E_0 is encountered during the isogeny walk.

In practice, E_0 is unlikely to ever be encountered in a random isogeny walk, except as the starting curve in the first phase of a key exchange, thus this flaw seems hard to exploit. Nevertheless, we find it not significantly more expensive to use a different approach, also suggested in [3]: with $u \neq 0$, only on E_0, we define the output of Elligator as $T_+ = (u : 1), T_- = (-u : 1)$ when $u^3 + u$ is a square, and we swap the points when $u^3 + u$ is not a square.

With these choices, under reasonable heuristics experimentally verified in [3], the running time of the whole algorithm is uncorrelated to the secret key as long as the values of u are unknown to an adversary. We summarize our implementation of Elligator in Algorithm 3, generalizing it to the case of Montgomery curves represented by projective coefficients (see also Section 4.1.1).

3.2 Fixing a leaking branch in Onuki–Aikawa–Yamazaki–Takagi

The algorithm from [26], essentially reproduced in Algorithm 2, includes a conditional statement at Line 12 which branches on the value of the point Q computed at Line 10. But this value depends on the sign s of the secret exponent e_i, so the branch leaks information about the secret. We propose repairing this by always computing both $Q_0 \leftarrow [k/\ell_i]P_0$ and $Q_1 \leftarrow [k/\ell_i]P_1$ at Line 10, and replacing the condition in Line 12 with a test for $(Q_0 = \infty)$ **or** $(Q_1 = \infty)$ (and using constant-time conditional swaps throughout).[8] This fix is visible in Line 13 of Algorithm 5.

[8] We also found a branch on secret data in the code provided with [21] at `https://zenon.cs.hs-rm.de/pqcrypto/faster-csidh`, during the 3-isogeny computation, when computing $[\ell]P = [(\ell - 1)/2]P + [(\ell + 1)/2]P$. This can be easily fixed by a conditional swap, without any significant impact on running time.

Algorithm 3: Constant-time projective Elligator

Input: A supersingular curve $E_{(A':C')} : C'y^2 = C'x^3 + A'x^2 + C'x$ over $\mathbb{F}_p$, and an element $u \in \{2, \ldots, \frac{p-1}{2}\}$.

Output: A pair of points $T_+ \in E_{(A':C')}[\pi - 1]$ and $T_- \in E_{(A':C')}[\pi + 1]$.

$t \leftarrow A'\big((u^2-1)u^2{A'}^2C' + ((u^2-1)C')^3\big)$

$a \leftarrow \texttt{isequal}(t, 0)$ // $t = 0$ `iff` $A' = 0$

$\alpha, \beta \leftarrow 0, u$

$\texttt{cswap}(\alpha, \beta, a)$ // $\alpha = 0$ `iff` $A' \neq 0$

$t' \leftarrow t + \alpha(u^2 + 1)$ // $t' \neq 0$

$T_+ \leftarrow (A' + \alpha C'(u^2-1) : C'(u^2-1))$

$T_- \leftarrow (-A'u^2 - \alpha C'(u^2-1) : C'(u^2-1))$

$b \leftarrow \texttt{Legendre\_symbol}(t', p)$ // $b = \pm 1$

$c \leftarrow \texttt{isequal}(b, -1)$

$\texttt{cswap}(T_+, T_-, c)$

return (T_+, T_-)

4 Optimizing constant-time implementations

In this section we propose several optimizations that are compatible with both non-constant-time and constant-time implementations of CSIDH.

4.1 Isogeny and point arithmetic on twisted Edwards curves

In this subsection, we present efficient formulas in twisted-Edwards coordinates for four fundamental operations: point addition, point doubling, isogeny computation (as presented in [25]; cf. §2.2), and isogeny evaluation (*i.e.* computing the image of a point under an isogeny). Our approach obtains a modest but still noticeable improvement with respect to previous proposals based on Montgomery representation, or hybrid strategies that propound combinations of Montgomery and twisted-Edwards representations [5,18,19,20,23].

Castryck, Galbraith, and Farashahi [5] proposed using a hybrid representation to reduce the cost of point doubling on certain Montgomery curves, by exploiting the fact that converting between Montgomery and twisted Edwards models can be done at almost no cost. In [23], Meyer, Reith and Campos considered using twisted Edwards formulas for computing isogeny and elliptic curve arithmetic, but concluded that a pure twisted-Edwards-only approach would not be advantageous in the context of SIDH. Bernstein, Lange, Martindale, and Panny observed in [3] that the conversion from Montgomery XZ coordinates to twisted Edwards YZ coordinates occurs naturally during the Montgomery ladder. Kim, Yoon, Kwon, Park, and Hong presented a hybrid model in [19] using Edwards and Montgomery modelds for isogeny computations and point arithmetic, respectively; in [18] and [20], they suggested computing isogenies using a modified twisted Edwards representation that introduces a fourth coordinate w.

To the best of our knowledge, the quest for more efficient elliptic curve and isogeny arithmetic than that offered by pure Montgomery and twisted-Edwards-

Montgomery representations remains an open problem. As a step forward in this direction, Moody and Shumow [25] showed that when dealing with isogenies of odd degree $d = 2\ell - 1$ with $\ell \geq 2$, twisted Edwards representation offers a cheaper formulation for isogeny computation than the corresponding one using Montgomery curves; nevertheless, they did not address the problem of getting a cheaper twisted Edwards formulation for the isogeny evaluation operation.

4.1.1 Montgomery curves A Montgomery curve [24] is defined by the equation $E_{A,B} : By^2 = x^3 + Ax^2 + x$, such that $B \neq 0$ and $A^2 \neq 4$ (we often write E_A for $E_{A,1}$). We refer to [9] for a survey on Montgomery curves. When performing isogeny computations and evaluations, it is often more convenient to represent the constant A in the projective space $\mathbb{P}^1$ as $(A' : C')$, such that $A = A'/C'$. Montgomery curves are attractive because they are exceptionally well-suited to performing the differential point addition operation which computes $x(P + Q)$ from $x(P)$, $x(Q)$, and $x(P - Q)$. Equations (1) and (2) describe the differential point doubling and addition operations proposed by Montgomery in [24]:

$$\begin{aligned} X_{[2]P} &= C_{24}(X_P + Z_P)^2(X_P - Z_P)^2, \\ Z_{[2]P} &= ((X_P + Z_P)^2 - (X_P - Z_P)^2)\cdot \\ &\qquad (C_{24}(X_P - Z_P)^2 + A_{24p}((X_P + Z_P)^2 - (X_P - Z_P)^2)) \end{aligned} \tag{1}$$

where $A_{24p} = A + 2C$ and $C_{24} = 4C$, and

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

Montgomery curves can be used to efficiently compute isogenies using Vélu's formulas [30]. Suppose we want the image of a point Q under an ℓ-isogeny ϕ, where $\ell = 2k + 1$. For each $1 \leq i \leq k$ we let $(X_i : Z_i) = x([i]P)$, where $\langle P \rangle = \ker \phi$. Equation (3) computes $(X' : Z') = x(\phi(Q))$ from $(X_Q : Z_Q) = x(Q)$.

$$\begin{aligned} X' &= X_P\Big(\prod_{i=1}^{k}\left[(X_Q - Z_Q)(X_i + Z_i) + (Z_Q + Z_Q)(X_i - Z_i)\right]\Big)^2 \\ Z' &= Z_P\Big(\prod_{i=1}^{k}\left[(X_Q - Z_Q)(X_i + Z_i) - (Z_Q + Z_Q)(X_i - Z_i)\right]\Big)^2 \end{aligned} \tag{3}$$

4.1.2 Twisted Edwards curves In [1] we see that every Montgomery curve $E_{A,B} : By^2 = x^3 + Ax^2 + x$ is birationally equivalent to a twisted Edwards curve $E_{a,d} : ax^2 + y^2 = 1 + dx^2y^2$; the curve constants are related by

$$(A, B) = \left(\frac{2(a+d)}{a-d}, \frac{4}{a-d}\right) \quad \text{and} \quad (a, d) = \left(\frac{A+2}{B}, \frac{A-2}{B}\right),$$

and the rational maps $\phi : E_{a,d} \to E_{A,B}$ and $\psi : E_{A,B} \to E_{a,d}$ are defined by

$$\begin{aligned} \phi &: (x, y) \longmapsto ((1+y)/(1-y), (1+y)/(1-yx)) \,, \\ \psi &: (x, y) \longmapsto (x/y, (x-1)/(x+1)) \,. \end{aligned} \tag{4}$$

Rewriting this relationship for Montgomery curves with projective constants, $E_{a,d}$ is equivalent to the Montgomery curve $E_{(A:C)} = E_{A/C,1}$ with constants

$$A_{24p} := A + 2C = a \,, \qquad A_{24m} := A - 2C = d \,, \qquad C_{24} := 4C = a - d \,.$$

To avoid notational ambiguities, we write $(Y_P : T_P)$ for the $\mathbb{P}^1$ projection of the y-coordinate of the point $P \in E_{a,d}$. Let $P \in E_{(A:C)}$. In projective coordinates, the map ψ of (4) becomes

$$\psi : (X_P : Z_P) \longmapsto (Y_P : T_P) = (X_P - Z_P : X_P + Z_P) \tag{5}$$

Comparing (5) with (1) reveals that Y_P and T_P appear in the doubling formula, so we can substitute them at no cost. Replacing A_{24p} and C_{24} with their twisted Edwards equivalents a and $e = a - d$, respectively, we obtain a doubling formula for twisted Edwards YT coordinates:

$$\begin{aligned} Y_{[2]P} &= e \cdot Y_P^2 \cdot T_P^2 - (T_P^2 - Y_P^2) \cdot (eY_P^2 + a(T_P^2 - Y_P^2)), \\ T_{[2]P} &= e \cdot Y_P^2 \cdot T_P^2 + (T_P^2 - Y_P^2) \cdot (eY_P^2 + a(T_P^2 - Y_P^2)). \end{aligned}$$

Similarly, the coordinates $Y_P, T_P, Y_Q, T_Q, Y_{P-Q}$ and T_{P-Q} appear in (2), and thus we derive differential addition formulas for twisted Edwards coordinates:

$$\begin{aligned} Y_{P+Q} &= (T_{P-Q} - Y_{P-Q})(Y_P T_Q + Y_Q Z_P)^2 - (T_{P-Q} + Y_{P-Q})(Y_P T_Q - Y_Q Z_P)^2, \\ T_{P+Q} &= (T_{P-Q} - Y_{P-Q})(Y_P T_Q + Y_Q Z_P)^2 + (T_{P-Q} + Y_{P-Q})(Y_P T_Q - Y_Q Z_P)^2. \end{aligned}$$

The computational costs of doubling and differential addition are $4\mathbf{M} + 2\mathbf{S} + 4\mathbf{A}$ (the same as evaluating (1)) and $4\mathbf{M} + 2\mathbf{S} + 6\mathbf{A}$ (the same as (2)), respectively.

The Moody–Shumow formulas for isogeny computation [25] are given in terms of twisted Edwards YT-coordinates. It remains to derive a twisted Edwards YT-coordinate isogeny-evaluation formula for ℓ-isogenies where $\ell = 2k+1$. We do this by applying the map in (5) to (3), which yields

$$\begin{aligned} Y' = {} & (T_{P-Q} + Y_{P-Q}) \cdot \Big(\prod_{i=1}^{k} \left[T_Q Y_{[i]P} + Y_Q T_{[i]P} \right] \Big)^2 \\ & - (T_{P-Q} - Y_{P-Q}) \cdot \Big(\prod_{i=1}^{k} \left[T_Q Y_{[i]P} - Y_Q T_{[i]P} \right] \Big)^2 , \\ T' = {} & (T_{P-Q} + Y_{P-Q}) \cdot \Big(\prod_{i=1}^{k} \left[T_Q Y_{[i]P} + Y_Q T_{[i]P} \right] \Big)^2 \\ & + (T_{P-Q} - Y_{P-Q}) \cdot \Big(\prod_{i=1}^{k} \left[T_Q Y_{[i]P} - Y_Q T_{[i]P} \right] \Big)^2 . \end{aligned}$$

The main advantage of the approach outlined here is that by only using points given in YT coordinates, we can compute point doubling, point addition and isogeny construction and evaluation at a lower computational cost. Indeed, isogeny evaluation in XZ costs $4k\mathbf{M} + 2\mathbf{S} + 6k\mathbf{A}$, whereas the above YT coordinate formula costs $4k\mathbf{M} + 2\mathbf{S} + (2k + 4)\mathbf{A}$, thus saving $4k - 4$ field additions.

4.2 Addition chains for a faster scalar multiplication

Since the coefficients in CSIDH scalar multiplications are always known in advance (they are essentially system parameters), there is no need to hide them by using constant-time scalar multiplication algorithms such as the classical Montgomery ladder. Instead, we can use shorter differential addition chains.[9]

In the CSIDH group action computation, any given scalar k is the product of a subset of the collection of the 74 small primes ℓ_i dividing $\frac{p+1}{4}$. We can take advantage of this structure to use shorter differential addition chains than those we might derive for general scalars of a comparable size. First, we pre-computed the shortest differential addition chains for each one of the small primes ℓ_i. One then computes the scalar multiplication operation $[k]P$ as the composition of the differential addition chains for each prime ℓ dividing k.

Power analysis on the coefficient computation might reveal the degree of the isogeny that is currently being computed, but, since we compute exactly one ℓ_i-isogeny for each ℓ_i per loop, this does not leak any secret information.

This simple trick allows us to compute scalar multiplications $[k]P$ using differential addition chains of length roughly $1.5\lceil \log_2(k) \rceil$. This yields a saving of about 25% compared with the cost of the classical Montgomery ladder.

5 Removing dummy operations for fault-attack resistance

The use of dummy operations in the previous constant-time algorithms implies that the attacker can obtain information on the secret key by injecting faults into variables during the computation. If the final result is correct, then she knows that the fault was injected in a dummy operation; if it is incorrect, then the operation was real. For example, if one of the values in in Line 18 of Algorithm 2 is modified without affecting the final result, then the adversary learns whether the corresponding exponent e_i was zero at that point.

Fault injection attacks have been considered in the context of SIDH ([15], [29]), but to the best of our knowledge, they have not been studied yet on dummy operations in the context of CSIDH. Below we propose an approach to constant-time CSIDH without dummy computations, making every computation essential for a correct final result. This gives us some natural resistance to fault, at the cost of approximately a twofold slowdown.

Our approach to avoiding fault-injection attacks is to change the format of secret exponent vectors $(e_1, \ldots, e_n)$. In both the original CSIDH and the Onuki

[9] A differential addition chain is an addition chain such that for every chain element c computed as $a + b$, the difference $a - b$ is already present in the chain.

et al. variants, the exponents e_i are sampled from an integer interval $[-m_i, m_i]$ centered in 0. For naive CSIDH, evaluating the action of $\mathfrak{l}_i^{e_i}$ requires evaluating between 0 and m isogenies, corresponding to either the ideal $\mathfrak{l}_i$ (for positive e_i) or $\mathfrak{l}_i^{-1}$ (for negative e_i). If we follow the approach of [26], then we must also compute $k - |e_i|$ dummy ℓ_i-isogenies to ensure a constant-time behaviour.

For our new algorithm, the exponents e_i are uniformly sampled from sets

$$\mathcal{S}(m_i) = \{e \mid e = m_i \bmod 2 \text{ and } |e| \leq m_i\},$$

i.e., centered intervals containing only even or only odd integers. The interesting property of these sets is that a vector drawn from $\mathcal{S}(m)^n$ can always be rewritten (in a non-unique way) as a sum of m vectors with entries $\{-1, +1\}$ (i.e., vectors in $\mathcal{S}(1)^n$). But the action of a vector drawn from $\mathcal{S}(1)^n$ can clearly be implemented in constant-time without dummy operations: for each coefficient e_i, we compute and evaluate the isogeny associated to $\mathfrak{l}_i$ if $e_i = 1$, or the one associated to $\mathfrak{l}_i^{-1}$ if $e_i = -1$. Thus, we can compute the action of vectors drawn from $\mathcal{S}(m)^n$ by repeating m times this step.

More generally, we want to evaluate the action of vectors $(e_1, \ldots, e_n)$ drawn from $\mathcal{S}(m_1) \times \cdots \times \mathcal{S}(m_n)$. Algorithm 4 achieves this in constant-time and without using dummy operations. The outer loop at line 3 is repeated exactly $\max(m_i)$ times, but the inner "if" block at line 5 is only executed m_i times for each i; it is clear that this flow does not depend on secrets. Inside the "if" block, the coefficients e_i are implicitly interpreted as

$$|e_i| = \underbrace{1 + 1 + \cdots + 1}_{e_i \text{ times}} + \underbrace{(1-1) - (1-1) + (1-1) - \cdots}_{m_i - e_i \text{ times}},$$

i.e., the algorithm starts by acting by $\mathfrak{l}_i^{\mathtt{sign}(e_i)}$ for e_i iterations, then alternates between $\mathfrak{l}_i$ and $\mathfrak{l}_i^{-1}$ for $m_i - e_i$ iterations. We assume that the $\mathtt{sign} : \mathbb{Z} \to \{\pm 1\}$ operation is implemented in constant time, and that $\mathtt{sign}(0) = 1$. If one is careful to implement the isogeny evaluations in constant-time, then it is clear that the full algorithm is also constant-time.

However, Algorithm 4 is only an idealized version of the CSIDH group action algorithm. Indeed, like in [21,26], it may happen in some iterations that Elligator outputs points of order not divisible by ℓ_i, and thus the action of $\mathfrak{l}_i$ or $\mathfrak{l}_i^{-1}$ cannot be computed in that iteration. In this case, we simply skip the loop and retry later: this translates into the variable z_i not being decremented, so the total number of iterations may end up being larger than $\max(m_i)$. Fortunately, if the input value u fed to Elligator is random, its output is uncorrelated to secret values[10], and thus the fact that an iteration is skipped does not leak information on the secret. The resulting algorithm is summarized in Algorithm 5.

To maintain the security of standard CSIDH, the bounds m_i must be chosen so that the key space is at least as large. For example, the original implementation [6] samples secrets in $[-5, 5]^{74}$, which gives a key space of size 11^{74}; hence,

[10] Assuming the usual heuristic assumptions on the distribution of the output of Elligator, see [21].

Algorithm 4: An idealized dummy-free constant-time evaluation of the CSIDH group action.

Input: Secret vector $(e_1, \ldots, e_n) \in \mathcal{S}(m_1) \times \cdots \times \mathcal{S}(m_n)$

$(t_1, \ldots, t_n) \leftarrow (\texttt{sign}(e_1), \ldots, \texttt{sign}(e_n))$ `// Secret`

$(z_1, \ldots, z_n) \leftarrow (m_1, \ldots, m_n)$ `// Not secret`

while *some* $z_i \neq 0$ **do**

  for $i \in \{1, \ldots, n\}$ **do**

    if $z_i > 0$ **then**

      Act by $\mathfrak{l}_i^{t_i}$

      $b = \texttt{isequal}(e_i, 0)$

      $e_i \leftarrow e_i - t_i$

      $t_i \leftarrow (-1)^b \cdot t_i$ `// Swap sign when` e_i `has gone past` 0

      $z_i \leftarrow z_i - 1$

to get the same security we would need to sample secrets in $\mathcal{S}(10)^{74}$. But a constant-time version of CSIDH *à la* Onuki *et al.* only needs to evaluate five isogeny steps per prime ℓ_i, whereas the present variant would need to evaluate ten isogeny steps. We thus expect an approximately twofold slowdown for this variant compared to Onuki *et al.*, which is confirmed by our experiments.

6 Derandomized CSIDH algorithms

As we stressed in Section 3, all of the algorithms presented here depend on the availability of high-quality randomness for their security. Indeed, the input to Elligator must be randomly chosen to ensure that the total running time is uncorrelated to the secret key. Typically, this would imply the use of a PRNG seeded with high quality true randomness that must be kept secret. An attack scenario where the attacker may know the output of the PRNG, or where the quality of PRNG output is less than ideal, therefore degrades the security of all algorithms. This is true even when the secret was generated with a high-quality PRNG if the keypair is static, and the secret key is then used by an algorithm with low-quality randomness.

We can avoid this issue completely if points of order $\prod \ell_i^{|m_i|}$, where $|m_i|$ is the maximum possible exponent (in absolute value) for ℓ_i, are available from the start. Unfortunately this is not possible with standard CSIDH, because such points are defined over field extensions of exponential degree.

Instead, we suggest modifying CSIDH as follows. First, we take a prime $p = 4\prod_{i=1}^{n} \ell_i - 1$ such that $\lceil n \log_2(3) \rceil = 2\lambda$, where λ is a security parameter, and we restrict to exponents of the private key sampled from $\{-1, 0, 1\}$. Then, we compute two points of order $(p+1)/4$ on the starting public curve, one in $\ker(\pi - 1)$ and the other in $\ker(\pi + 1)$, where π is the Frobenius endomorphism. This computation involves no secret information and can be implemented in variable-time; furthermore, if the starting curve is the initial curve with $A = 0$,

Algorithm 5: Dummy-free randomized constant-time CSIDH class group action for supersingular curves over $\mathbb{F}_p$, where $p = 4\prod_{i=1}^{n} \ell_i - 1$. The ideals $\mathfrak{l}_i = (\ell_i, \pi - 1)$, where π maps to the p-th power Frobenius endomorphism on each curve, and the vector $(m_1, \ldots, m_n)$ of exponent bounds, are system parameters. This algorithm computes exactly m_i isogenies for each ideal $\mathfrak{l}_i$.

Input: A supersingular curve E_A over $\mathbb{F}_p$, and an exponent vector $(e_1, \ldots, e_n)$ with each $e_i \in [-m_i, m_i]$ and $e_i \equiv m_i \pmod 2$.

Output: $E_B = \mathfrak{l}_1^{e_1} * \cdots * \mathfrak{l}_n^{e_n} * E_A$.

$(t_1, \ldots, t_n) \leftarrow \left(\frac{\mathtt{sign}(e_1)+1}{2}, \ldots, \frac{\mathtt{sign}(e_n)+1}{2}\right)$ `// Secret`

$(z_1, \ldots, z_n) \leftarrow (m_1, \ldots, m_n)$ `// Not secret`

$E_B \leftarrow E_A$

while *some* $z_i \neq 0$ **do**

  $u \leftarrow \mathtt{Random}\left(\{2, \ldots, \frac{p-1}{2}\}\right)$

  $(T_1, T_0) \leftarrow \mathtt{Elligator}(E_B, u)$ `//` $T_1 \in E_B[\pi - 1]$ `and` $T_0 \in E_B[\pi + 1]$

  $(T_0, T_1) \leftarrow ([4]T_0, [4]T_1)$ `// Now` $T_0, T_1 \in E_B\left[\prod_i \ell_i\right]$

  for $i \in \{1, \ldots, n\}$ **do**

    if $z_i \neq 0$ **then**

      $(G_0, G_1) \leftarrow (T_0, T_1)$

      for $j \in \{i + 1, \ldots, n\}$ **do**

        $(G_0, G_1) \leftarrow ([\ell_j]G_0, [\ell_j]G_1)$

      if $G_0 \neq \infty$ *and* $G_1 \neq \infty$ **then**

        $\mathtt{cswap}(G_0, G_1, t_i)$ `// Secret kernel point generator:` G_0

        $\mathtt{cswap}(T_0, T_1, t_i)$ `// Secret point to be multiplied:` T_1

        $(E_B, \phi) \leftarrow \mathtt{QuotientIsogeny}(E_B, G_0)$

        $(T_0, T_1) \leftarrow \big(\phi(T_0), \phi(T_1)\big)$

        $T_1 \leftarrow [\ell_i]T_1$

        $\mathtt{cswap}(T_0, T_1, t_i)$

        $b \leftarrow \mathtt{isequal}(e_i, 0)$

        $e_i \leftarrow e_i + (-1)^{t_i}$

        $t_i \leftarrow t_i \oplus b$

        $z_i \leftarrow z_i - 1$

      else if $G_0 \neq \infty$ **then**

        $T_0 \leftarrow [\ell_i]T_0$

      else if $G_1 \neq \infty$ **then**

        $T_1 \leftarrow [\ell_i]T_1$

return B

or a public curve corresponding to a long term secret key, these points can be precomputed offline and attached to the system parameters or the public key. We also remark that even for ephemeral public keys, a point of order $p+1$ must be computed anyway for key validation purposes, and thus this computation only slows down key validation by a factor of two.

Since we have restricted exponents to $\{-1, 0, 1\}$, every ℓ_i-isogeny in Algorithm 2 can be computed using only (the images of) the two precomputed points. There is no possibility of failure in the test of Line 12, and no need to sample any other point.

We note that this algorithm still uses dummy operations. If fault-injection attacks are a concern, the exponents can be further restricted to $\{-1, 1\}$, and the group action evaluated as in (a stripped down form of) Algorithm 5. However this further increases the size of p, as n must now be equal to 2λ.

This protection comes at a steep price: at the 128 bits security level, the prime p goes from 511 bits to almost 1500. The resulting field arithmetic would be considerably slower, although the global running time would be slightly offset by the smaller number of isogenies to evaluate.

On the positive side, the resulting system would have much stronger quantum security. Indeed, the best known quantum attacks are exponential in the size of the key space ($\approx 2^{2\lambda}$ here), but only subexponential in p (see [7,13,6]). Since our modification more than doubles the size of p without changing the size of the key space, quantum security is automatically increased. For this same reason, for security levels beyond NIST-1 (64 quantum bits of security), the size of p increases more than linearly in λ, and the variant proposed here becomes natural. Finally, parameter sets with a similar imbalance between the size of p and the security parameter λ have already been considered in the context of isogeny based signatures [11], where they provide tight security proofs in the QROM.

Hence, while at the moment this costly modification of CSIDH may seem overkill, we believe further research is necessary to try and bridge the efficiency gap between it and the other side-channel protected implementations of CSIDH.

7 Experimental results

Tables 1 and 2 summarize our experimental results, and compare our algorithms with those of [6], [21], and [26]. Table 1 compares algorithms in terms of elementary field operations, while Table 2 compares cycle counts of C implementations. All of our experiments were ran on a Intel(R) Core(TM) i7-6700K CPU 4.00GHz machine with 16GB of RAM. Turbo boost was disabled. The software environment was the Ubuntu 16.04 operating system and `gcc` version 5.5.

In all of the algorithms considered here (except the original [6]), the group action is evaluated using the SIMBA method (Splitting Isogeny computations into Multiple BAtches) proposed by Meyer, Campos, and Reith in [21]. Roughly speaking, SIMBA-m-k partitions the set of primes ℓ_i into m disjoint subsets S_i (batches) of approximately the same size. SIMBA-m-k proceeds by computing

isogenies for each batch S_i; after k steps, the unreached primes ℓ_i from each batch are merged.

Castryck et al. We used the reference CSIDH implementation made available for download by the authors of [6]. None of our countermeasures or algorithmic improvements were applied.

Meyer–Campos–Reith. We used the software library freely available from the authors of [21]. This software batches isogenies using SIMBA-5-11. The improvements we describe in §3 and §4 were *not* applied.

Onuki et. al. Unfortunately, the source code for the implementation in [26] was not freely available, so direct comparison with our implementation was impossible. Table 1 includes their field operation counts for their unmodified algorithm (which, as noted in §3, is insecure) using SIMBA-3-8, and our estimates for a repaired version applying our fix in §3. We did not apply the optimizations of §4 here. (We do not replicate the cycle counts from [26] in Table 2, since they may have been obtained using turbo boost, thus rendering any comparison invalid.)

Our implementations. We implemented three constant-time CSIDH algorithms, using the standard primes with the exponent bounds m_i from [26, §5.2].

MCR-style This is essentially our version of Meyer–Campos–Reith (with one torsion point and dummy operations, batching isogenies with SIMBA-5-11), but applying the techniques of §3 and §4.

OAYT-style This is essentially our version of Onuki *et. al.* (using two torsion points and dummy operations, batching isogenies with SIMBA-3-8), but applying the techniques of §3 and §4.

No-dummy This is Algorithm 5 (with two torsion points and no dummy operations), batching isogenies using SIMBA-5-11.

In each case, the improvements and optimizations of §3-4 are applied, including projective Elligator, short differential addition chains, and twisted Edwards arithmetic and isogenies. Our software library is freely available from

`https://github.com/JJChiDguez/csidh`.

The field arithmetic is based on the Meyer–Campos–Reith software library [21]; since the underlying arithmetic is essentially identical, the performance comparisons below reflect differences in the CSIDH algorithms.

Results. We see in Table 2 that the techniques we introduced in §3 and §4 produce substantial savings compared with the implementation of [21]. In particular, our OAYT-style implementation yields a 25% improvement over [21]. Since the implementations use the same underlying field arithmetic library, these improvements are entirely due to the techniques introduced in this paper. While our no-dummy variant is (unsurprisingly) slower, we see that the performance penalty is not prohibitive: it is less than twice as slow as our fastest dummy-operation algorithm, and only 44% slower than [21].

Table 1. Field operation counts for constant-time CSIDH. Counts are given in millions of operations, averaged over 1024 random experiments. The counts for a possible repaired version of [26] are estimates, and hence displayed in italics. The performance ratio uses [21] as a baseline, considers only multiplication and squaring operations, and assumes $M = S$.

Implementation	CSIDH Algorithm	M	S	A	Ratio
Castryck et al. [6]	unprotected, unmodified	0.252	0.130	0.348	0.26
Meyer–Campos–Reith [21]	unmodified	1.054	0.410	1.053	1.00
Onuki et al. [26]	unmodified	0.733	0.244	0.681	0.67
	repaired as in §3	*0.920*	*0.338*	*0.867*	*0.86*
This work	MCR-style	0.901	0.309	0.965	0.83
	OAYT-style	0.802	0.282	0.900	0.74
	No-dummy	1.525	0.526	1.686	1.40

Table 2. Clock cycle counts for constant-time CSIDH implementations, averaged over 1024 experiments. The ratio is computed using [21] as baseline implementation.

Implementation	CSIDH algorithm	Mcycles	Ratio
Castryck et al. [6]	unprotected, unmodified	155	0.39
Meyer–Campos–Reith [21]	unmodified	395	1.00
This work	MCR-style	337	0.85
	OAYT-style	300	0.76
	No-dummy	569	1.44

8 Conclusion and perspectives

We studied side-channel protected implementations of the isogeny based primitive CSIDH. Previous implementations failed at being constant time because of some subtle mistakes. We fixed those problems, and proposed new improvements, to achieve the most efficient version of CSIDH protected against timing and simple power analysis attacks to date. All of our algorithms were implemented in C, and the source made publicly available online.

We also studied the security of CSIDH in stronger attack scenarios. We proposed a protection against some fault-injection and timing attacks that only comes at a cost of a twofold slowdown. We also sketched an alternative version of CSIDH "for the paranoid", with much stronger security guarantees, however at the moment this version seems too costly for the security benefits; more work is required to make it competitive with the original definition of CSIDH.

References

1. Daniel J. Bernstein, Peter Birkner, Marc Joye, Tanja Lange, and Christiane Peters. Twisted Edwards curves. In Serge Vaudenay, editor, *Progress in Cryptology - AFRICACRYPT 2008*, volume 5023 of *Lecture Notes in Computer Science*, pages 389–405. Springer, 2008.

2. Daniel J. Bernstein, Mike Hamburg, Anna Krasnova, and Tanja Lange. Elligator: elliptic-curve points indistinguishable from uniform random strings. In *2013 ACM SIGSAC Conference on Computer and Communications Security, CCS'13, Berlin, Germany, November 4-8, 2013*, pages 967–980, 2013.
3. Daniel J. Bernstein, Tanja Lange, Chloe Martindale, and Lorenz Panny. Quantum circuits for the CSIDH: optimizing quantum evaluation of isogenies. In *Advances in Cryptology - EUROCRYPT 2019 - 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, May 19-23, 2019, Proceedings, Part II*, pages 409–441, 2019.
4. Ward Beullens, Thorsten Kleinjung, and Frederik Vercauteren. CSI-FiSh: Efficient isogeny based signatures through class group computations. *IACR Cryptology ePrint Archive*, 2019:498, 2019.
5. Wouter Castryck, Steven D. Galbraith, and Reza Rezaeian Farashahi. Efficient arithmetic on elliptic curves using a mixed Edwards–Montgomery representation. *Cryptology ePrint Archive*, 2008:218, 2008.
6. Wouter Castryck, Tanja Lange, Chloe Martindale, Lorenz Panny, and Joost Renes. CSIDH: an efficient post-quantum commutative group action. In *Advances in Cryptology - ASIACRYPT 2018 - 24th International Conference on the Theory and Application of Cryptology and Information Security, Brisbane, QLD, Australia, December 2-6, 2018, Proceedings, Part III*, pages 395–427, 2018.
7. Andrew M. Childs, David Jao, and Vladimir Soukharev. Constructing elliptic curve isogenies in quantum subexponential time. *J. Mathematical Cryptology*, 8(1):1–29, 2014.
8. Craig Costello, Patrick Longa, and Michael Naehrig. Efficient algorithms for supersingular isogeny Diffie–Hellman. In *Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part I*, pages 572–601, 2016.
9. Craig Costello and Benjamin Smith. Montgomery curves and their arithmetic - the case of large characteristic fields. *J. Cryptographic Engineering*, 8(3):227–240, 2018.
10. Jean Marc Couveignes. Hard homogeneous spaces. Cryptology ePrint Archive, Report 2006/291, 2006.
11. Luca De Feo and Steven D. Galbraith. SeaSign: Compact isogeny signatures from class group actions. Cryptology ePrint Archive, Report 2018/824, 2018.
12. Luca De Feo, David Jao, and Jérôme Plût. Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies. *J. Mathematical Cryptology*, 8(3):209–247, 2014.
13. Luca De Feo, Jean Kieffer, and Benjamin Smith. Towards practical key exchange from ordinary isogeny graphs. In *Advances in Cryptology - ASIACRYPT 2018 - 24th International Conference on the Theory and Application of Cryptology and Information Security, Brisbane, QLD, Australia, December 2-6, 2018, Proceedings, Part III*, pages 365–394, 2018.
14. Thomas Decru, Lorenz Panny, and Frederik Vercauteren. Faster SeaSign signatures through improved rejection sampling. to appear at PQCrypto 2019, 2019.
15. Alexandre Gélin and Benjamin Wesolowski. Loop-abort faults on supersingular isogeny cryptosystems. In *Post-Quantum Cryptography - 8th International Workshop, PQCrypto 2017, Utrecht, The Netherlands, June 26-28, 2017, Proceedings*, pages 93–106, 2017.
16. Amir Jalali, Reza Azarderakhsh, Mehran Mozaffari Kermani, and David Jao. Towards optimized and constant-time CSIDH on embedded devices. In *Constructive*

Side-Channel Analysis and Secure Design, pages 215–231. Springer International Publishing, 2019.
17. David Jao and Luca De Feo. Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies. In *Post-Quantum Cryptography - 4th International Workshop, PQCrypto 2011, Taipei, Taiwan, November 29 - December 2, 2011. Proceedings*, pages 19–34, 2011.
18. Suhri Kim, Kisoon Yoon, Jihoon Kwon, Seokhie Hong, and Young-Ho Park. Efficient isogeny computations on twisted Edwards curves. *Security and Communication Networks*, 2018.
19. Suhri Kim, Kisoon Yoon, Jihoon Kwon, Young-Ho Park, and Seokhie Hong. New hybrid method for isogeny-based cryptosystems using Edwards curves. Cryptology ePrint Archive, Report 2018/1215, 2018. https://eprint.iacr.org/2018/1215.
20. Suhri Kim, Kisoon Yoon, Young-Ho Park, and Seokhie Hong. Optimized method for computing odd-degree isogenies on Edwards curves. Cryptology ePrint Archive, Report 2019/110, 2019. https://eprint.iacr.org/2019/110.
21. Michael Meyer, Fabio Campos, and Steffen Reith. On lions and elligators: An efficient constant-time implementation of CSIDH. In *Post-Quantum Cryptography - 10th International Workshop, PQCrypto 2019*, 2019.
22. 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*, pages 137–152, 2018.
23. Michael Meyer, Steffen Reith, and Fabio Campos. On hybrid SIDH schemes using Edwards and Montgomery curve arithmetic. *Cryptology ePrint Archive*, 2017:1213, 2017.
24. Peter L. Montgomery. Speeding the Pollard and elliptic curve methods of factorization. *Mathematics of Computation*, 48:243–234, 1987.
25. Dustin Moody and Daniel Shumow. Analogues of Vélu's formulas for isogenies on alternate models of elliptic curves. *Mathematics of Computation*, 85(300):1929–1951, 2016.
26. Hiroshi Onuki, Yusuke Aikawa, Tsutomu Yamazaki, and Tsuyoshi Takagi. A faster constant-time algorithm of CSIDH keeping two torsion points. To appear in IWSEC 2019 – The 14th International Workshop on Security, 2019.
27. Alexander Rostovtsev and Anton Stolbunov. Public-key cryptosystem based on isogenies. Cryptology ePrint Archive, Report 2006/145, 2006.
28. Anton Stolbunov. Constructing public-key cryptographic schemes based on class group action on a set of isogenous elliptic curves. *Advances in Mathematics of Communication*, 4(2), 2010.
29. Yan Bo Ti. Fault attack on supersingular isogeny cryptosystems. In *Post-Quantum Cryptography - 8th International Workshop, PQCrypto 2017, Utrecht, The Netherlands, June 26-28, 2017, Proceedings*, pages 107–122, 2017.
30. Jacques Vélu. Isogénies entre courbes elliptiques. *Comptes-rendu de l'académie des sciences de Paris*, 1971.