Disorientation faults in CSIDH

Gustavo Banegas[1], Juliane Krämer[2], Tanja Lange[3,4], Michael Meyer[2], Lorenz Panny[4], Krijn Reijnders[5], Jana Sotáková[6], and Monika Trimoska[5]

[1] Inria and Laboratoire d'Informatique de l'Ecole polytechnique, Institut Polytechnique de Paris, Palaiseau, France
`gustavo@cryptme.in`
[2] University of Regensburg, Germany
`juliane.kraemer@ur.de, michael@random-oracles.org`
[3] Eindhoven University of Technology, the Netherlands
`tanja@hyperelliptic.org`
[4] Academia Sinica, Taipei, Taiwan
`lorenz@yx7.cc`
[5] Radboud University, Nijmegen, The Netherlands
`krijn@cs.ru.nl, monika.trimoska@ru.nl`
[6] University of Amsterdam and QuSoft, Amsterdam, The Netherlands
`j.s.sotakova@uva.nl`

Abstract. We investigate a new class of fault-injection attacks against the CSIDH family of cryptographic group actions. Our *disorientation attacks* effectively flip the direction of some isogeny steps. We achieve this by faulting a specific subroutine, connected to the Legendre symbol or Elligator computations performed during the evaluation of the group action. These subroutines are present in almost all known CSIDH implementations. Post-processing a set of faulty samples allows us to infer constraints on the secret key. The details are implementation specific, but we show that in many cases, it is possible to recover the full secret key with only a modest number of successful fault injections and modest computational resources. We provide full details for attacking the original CSIDH proof-of-concept software as well as the CTIDH constant-time implementation. Finally, we present a set of lightweight countermeasures against the attack and discuss their security.

[*] Author list in alphabetical order; see https://ams.org/profession/leaders/CultureStatement04.pdf. This work began at the online Lorentz Center workshop "Post-Quantum Cryptography for Embedded Systems" held in February 2022. This research was funded in part by the European Commission through H2020 SPARTA (https://sparta.eu/), the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy—EXC 2092 CASA—390781972 "Cyber Security in the Age of Large-Scale Adversaries", the Taiwan's Executive Yuan Data Safety and Talent Cultivation Project (AS-KPQ-109-DSTCP), the German Federal Ministry of Education and Research (BMBF) under the project Quantum-RISC (ID 16KIS1039), the Dutch Research Council (NWO) through Gravitation-grant Quantum Software Consortium – 024.003.037, and a gender balance subsidy of the Faculty of Science, Radboud University, project number 6201362. This work was done in part while Tanja Lange was visiting the Simons Institute for the Theory of Computing. Date of this document: 2022-09-12.

Keywords: Fault-injection attack · isogenies of elliptic curves · post-quantum cryptography

1 Introduction

Isogeny-based cryptography is a contender in the ongoing quest for post-quantum cryptography. Perhaps the most attractive feature is small key size, but there are other reasons in favor of isogenies: Some functionalities appear difficult to construct from other paradigms. For instance, the *CSIDH* [15] scheme gives rise to non-interactive key exchange. CSIDH uses the action of an ideal-class group on a set of elliptic curves to mimic (some) classical constructions based on discrete logarithms, most notably the Diffie–Hellman key exchange. Recently, more advanced cryptographic protocols have been proposed based on the CSIDH group action: the signature schemes SeaSign [23] and CSI-FiSh [8], threshold schemes [24], oblivious transfer [27], and more. The main drawback of isogeny-based cryptography is speed: CSIDH takes hundreds of times longer to complete a key exchange than pre-quantum elliptic-curve cryptography (ECC).

The group action in CSIDH and related schemes is evaluated by computing a sequence of small-degree *isogeny steps*; the choice of degrees and "directions" is the private key. Thus, the control flow of a straightforward implementation is directly related to the secret key, which complicates side-channel resistant implementations [3, 7, 12, 26, 30, 31].

In a side-channel attack, passive observations of physical leakage (such as timing differences, electromagnetic emissions, or power consumption) during the execution of sensitive computations help an attacker infer secret information. A more intrusive class of physical attacks are *fault-injection attacks* or *fault attacks*: By actively manipulating the execution environment of a secure device (for instance, by altering the characteristics of the power supply, or by exposing the device to electromagnetic radiation), the attacker aims to trigger an error during the execution of sensitive computations and later infer secret information from the now incorrect, *faulty* outputs.

Two major classes of faults are *instruction skips* and *variable modifications*. Well-timed skips of processor instructions can have far-reaching consequences, for example, omitting a security check entirely, or failing to erase secrets which subsequently leak into the output. Depending on the attack model, variable modifications may reach from simply randomized CPU registers to precisely targeted single-bit flips. They cause the software to operate on unexpected values, which (especially in a cryptographic context) may lead to exploitable behavior. In practice, the difficulty of injecting a particular kind of fault (or combination of multiple faults) depends on various parameters; generally speaking, less targeted faults are easier.

Our contributions. We analyze the behavior of existing CSIDH implementations under a new class of attacks that we call *disorientation faults*. These faults occur when the attacker confuses the algorithm about the *orientation* of a point used during the computation: The effect of such an error is that a subset of the secret-dependent isogeny steps will be performed in the opposite direction, resulting in an incorrect output curve.

The placement of the disorientation fault during the algorithm influences the distribution of the output curve in a key-dependent manner. We explain how an attacker can post-process a set of faulty outputs to fully recover the private key. This attack works against almost all existing CSIDH implementations.

To simplify exposition we first assume access to a device that applies a secret key to a given public key (i.e., computing the shared key in CSIDH) and returns the result (for instance a hardware security module providing a CSIDH accelerator). We also discuss variants of the attack with weaker access; this includes a *hashed* version where faulty outputs are not revealed as-is, but passed through a key-derivation function first, as is commonly done for a Diffie–Hellman-style key exchange, and made available to the attacker only indirectly, e.g., as a MAC under the derived key.

Part of the tooling for the post-processing stage of our attack is a somewhat optimized meet-in-the-middle *path-finding* program for the CSIDH isogeny graph, dubbed `pubcrawl`. This software is intentionally kept fully generic with no restrictions specific to the fault-attack scenario we are considering, so that it may hopefully be usable out of the box for other applications requiring "small" neighborhood searches in CSIDH in the future.

Applying expensive but feasible precomputation can speed up post-processing for all attack variants and is particularly beneficial to the hashed version of the attack.

To defend against disorientation faults, we provide a set of *countermeasures*. We show different forms of protecting an implementation and discuss the pros and cons of each of the methods. In the end, we detail two of the protections that we believe give the best security. Both of them are lightweight, and they do not significantly add to the complexity of the implementation.

Related work. Prior works investigating fault attacks on isogeny-based cryptography mostly target specific variants or implementations of schemes and are different from our approach.

Loop-abort faults on the SIDH cryptosystem [25], discussed for CSIDH in [10], lead to leakage of an intermediate value of the computation rather than the final result. Replacing torsion points with other points in SIDH [36, 37] can be used to recover the secret keys; faulting intermediate curves in SIDH [2] to learn if secret isogeny paths lead over subfield curves can also leak information on secret keys. But the two latter attacks cannot be mounted against CSIDH due to the structural and mathematical differences between SIDH and CSIDH.

Recently, several CSIDH-specific fault attacks were published. One can modify memory locations and observe if this changes the resulting shared secret [11]. A different attack venue is to target fault injections against dummy computations in CSIDH [10, 28]. We emphasize that these are attacks against specific implementations and variants of CSIDH. To the best of our knowledge, our work features the first generic fault attack exploiting an operation and data flow present in almost all current implementations of CSIDH.

1.1 Note on security

We emphasize that CSIDH, its variants, and the protocols based on the CSIDH group action are not affected by the recent attacks that break the isogeny-based scheme SIDH and its instantiation SIKE [14,29,34]. These attacks exploit specific auxiliary information which is revealed in SIDH but does not exist in CSIDH.

CSIDH is a relatively young cryptosystem, being introduced only in 2018, but it is based on older systems due to Couveignes [21] and Rostovtsev and Stolbunov [35] which have received attention since 2006. The best non-quantum attack is a meet-in-the-middle attack running in $O(\sqrt[4]{p})$; a low-memory version was developed by Delfs and Galbraith in [22]. On a large quantum computer Kuperberg's attack can be mounted as shown by Childs, Jao, and Soukharev in [19]. This attack runs in $L_{\sqrt{p}}(1/2)$ calls to a quantum oracle. The number of oracle calls was further analyzed in [9] and [33] for concrete parameters while [7] analyzes the costs per oracle call in number of quantum operations. Combining these results shows that breaking CSIDH-512 requires around 2^{60} qubit operations on logical qubits, i.e., not taking into account the overhead for quantum error correction. Implementation papers such as CTIDH [3] use the CSIDH-512 prime for comparison purposes and also offer larger parameters. Likewise, we use the CSIDH-512 and CTIDH-512 parameters for concrete examples.

2 Background

CSIDH [15] is based on a group action on a certain set of elliptic curves. We explain the setup of CSIDH in Section 2.1 and relevant algorithmic aspects in Section 2.2. We assume some familiarity with elliptic curves and isogenies; the reader may consult [15] for more details.

2.1 CSIDH

We fix a prime p of the form $p = 4 \cdot \ell_1 \cdots \ell_n - 1$ with distinct odd primes ℓ_i. We define $\mathcal{E}$ to be the set of supersingular elliptic curves over $\mathbb{F}_p$ in Montgomery form, up to $\mathbb{F}_p$-isomorphism. All such curves admit an equation of the form $E_A : y^2 = x^3 + Ax^2 + x$ with a unique $A \in \mathbb{F}_p$. For $E_A \in \mathcal{E}$, the group of rational points $E_A(\mathbb{F}_p)$ is cyclic of order $p+1$. The quadratic twist of $E_A \in \mathcal{E}$ is $E_{-A} \in \mathcal{E}$.

Isogeny steps. For any ℓ_i and any $E_A \in \mathcal{E}$ there are two ℓ_i-isogenies, each leading to another curve in $\mathcal{E}$. One has kernel generated by any point P_+ of order ℓ_i with both coordinates in $\mathbb{F}_p$. We say this ℓ_i-isogeny is in the *positive direction* and the point P_+ has *positive orientation.* The other ℓ_i-isogeny has kernel generated by any point P_- of order ℓ_i with x-coordinate in $\mathbb{F}_p$ but y-coordinate in

$\mathbb{F}_{p^2} \setminus \mathbb{F}_p$. We say this isogeny is in the *negative direction* and the point P_- has *negative orientation*. Replacing E_A by the codomain of a positive and negative ℓ_i-isogeny from E_A is a *positive and negative ℓ_i-isogeny step*, respectively. As the name suggests, a positive and a negative ℓ_i-isogeny step cancel.

Fix $i \in \mathbb{F}_{p^2} \setminus \mathbb{F}_p$ with $i^2 = -1 \in \mathbb{F}_p$ and note that a negatively oriented point is necessarily of the form (x, iy) with $x, y \in \mathbb{F}_p$. Moreover, $x \in \mathbb{F}_p^*$ defines a positively oriented point on E_A whenever $x^3 + Ax^2 + x$ is a square in $\mathbb{F}_p$, and a negatively oriented point otherwise.

The group action. It is a non-obvious, but extremely useful fact that the isogeny steps defined above *commute*: Any sequence of them can be rearranged arbitrarily without changing the final codomain curve [15].

Thus, taking a combination of various isogeny steps defines a group action of the abelian group $(\mathbb{Z}^n, +)$ on $\mathcal{E}$: The vector $(e_1, \ldots, e_n) \in \mathbb{Z}^n$ represents $|e_i|$ individual ℓ_i-isogeny steps, with the sign of e_i specifying the orientation. In other words, letting $\mathfrak{l}_i$ denote a single positive ℓ_i-isogeny step, acting by $(e_1, \ldots, e_n) \in \mathbb{Z}^n$ on a curve E encodes the sequence of steps:

$$(\mathfrak{l}_1^{e_1} \cdots \mathfrak{l}_n^{e_n}) * E\,.$$

We refer to $(e_1, \ldots, e_n)$ as an *exponent vector*.

2.2 Algorithmic aspects

Every step is an oriented isogeny, so applying a single $\mathfrak{l}_i^{\pm 1}$ step requires a point P on E with two properties: It must have order ℓ_i and the correct orientation. The codomain of the corresponding isogeny from P is computed using either the Vélu [39] or $\sqrt{\text{é}}$lu [5] formulas.

Determining orientations. All state-of-the-art implementations of CSIDH use x-only arithmetic and completely disregard y-coordinates. So, we typically sample a point P by sampling an x-coordinate in $\mathbb{F}_p$. To determine the orientation of P, we then find the field of definition of the y-coordinate, for instance through a Legendre symbol computation. An alternative method is given by the "Elligator 2" map [6] which generates a point of the desired orientation.

Sampling order-ℓ points. There are several methods to compute points of given order ℓ. The following Las Vegas algorithm is popular for its simplicity and efficiency: As above, sample a uniformly random point P of either positive or negative orientation, and compute $Q := [(p+1)/\ell]P$. Since P is uniformly random in a cyclic group of order $p+1$, the point Q has order ℓ with probability $1 - 1/\ell$. With probability $1/\ell$, we get $Q = \infty$. Retry until $Q \neq \infty$. Filtering for points of given orientation is straightforward.

Multiple isogenies from a single point. To amortize the cost of sampling points and determining orientations, implementations usually pick some set S of indices belonging to exponents of the same sign, and attempt to compute one isogeny per degree ℓ_i with $i \in S$ from one point. If $d = \prod_{i\in S} \ell_i$ and P a random point, then the point $Q = [\frac{p+1}{d}]P$ has order dividing d. We can multiply Q by d/ℓ_i to construct an isogeny step for $\ell_i \in S$. The image of Q under the isogeny has the same orientation as P and Q and order dividing d/ℓ_i, so we continue with the next ℓ_j.

In CSIDH and its variants, the set S of isogeny degrees depends on the secret key and the orientation s of P. For example in Algorithm 1, for the first point that is sampled with positive orientation, the set S is $\{i \mid e_i > 0\}$.

The order of a random point P is not divisible by ℓ_i with probability $1/\ell_i$. This means that in many cases, we will not be able to perform an isogeny for *every* $i \in S$, but only for some (large) subset $S' \subset S$ due to P lacking factors ℓ_i in its order for those remaining $i \in S \setminus S'$. In short, a point P performs the action $\prod_{i\in S'} \mathfrak{l}_i^s$ for some $S' \subset S$, with s the orientation of P (interpreted as ± 1).

Sampling a point and computing the action $\prod_{i\in S'} \mathfrak{l}_i^s$ is called a *round*; we perform rounds for different sets S until we compute the full action $\mathfrak{a} = \prod \mathfrak{l}_i^{e_i}$.

Strategies. There are several ways of computing the group action as efficiently as possible, usually referred to as *strategies*. The strategy in Algorithm 1 is called *multiplicative strategy* [7,15,31]. Other notable strategies from the literature are the *SIMBA strategy* [30], *point-pushing strategies* [18], and *atomic blocks* [3].

1-point and 2-point approaches. The approach above and in Algorithm 1 samples a single point, computes some isogenies with the same orientation, and repeats this until all steps $\mathfrak{l}_i^{\pm 1}$ are processed. This approach, introduced in [15], is called *1-point approach*. In contrast, one can sample two points per round, one with positive and one with negative orientation, and attempt to compute isogenies for each degree ℓ_i per round, independent of the sign of the e_i [32]. Constant-time algorithms require choosing S independent of the secret key, and all state-of-the-art constant-time implementations use the *2-point approach*, e.g., [3,17].

Keyspace. In both CSIDH and CTIDH, each party's private key is an integer vector $(e_1, \ldots, e_n)$ sampled from a bounded subset $\mathcal{K} \subset \mathbb{Z}^n$, the *keyspace*. Different choices of $\mathcal{K}$ have different performance and security properties.

The original scheme [15] uses a keyspace $\mathcal{K}_m = \{-m, \ldots, m\}^n \subset \mathbb{Z}^n$; for CSIDH-512 the bound is typically $m = 5$ [15]. As suggested in [15, Remark 14] and shown in [30], using different bounds m_i for each i can improve speed. The shifted keyspace $\mathcal{K}_m^+ = \{0, \ldots, 2m_i\}^n$ was used in [30]. Other choices of $\mathcal{K}$ were made in [16,17,32], and CTIDH [3] (see Section 5.2).

3 Attack scenario and fault model

Throughout this work, we assume physical access to some hardware device containing an unknown CSIDH private key $\mathfrak{a}$. In the basic version of the attack, we suppose that the device provides an interface to pass in a CSIDH public-key curve E and receive back the result $\mathfrak{a} * E$ of applying $\mathfrak{a}$ to the public key E as in the second step of the key exchange.

Remark 1. Diffie–Hellman-style key agreements typically *hash* the shared secret to derive symmetric key material, instead of directly outputting curves as in our scenario. Our attacks are still applicable in this *hashed version* of the attack, although the complexity for post-processing steps from Section 4 will increase significantly. To simplify exposition, we postpone this discussion to Section 7.

We assume that the attacker is able to trigger an error during the computation of the orientation of a point in a specific round of the CSIDH algorithm: whenever a point P with orientation $s \in \{-1, 1\}$ is sampled during the algorithm, we can flip the orientation $s \mapsto -s$ as shown below. This leads to some isogenies being computed in the opposite direction throughout the round. The effect of this flip will be explored in Section 4.

Square check. In CSIDH, cf. Algorithm 1, the point P is generated in Step 2 and its orientation s is determined in Step 3. The function `IsSquare` determines s by taking as input the non-zero value $z = x^3 + Ax^2 + x$, and computing the Legendre symbol of z. Hence, $s = 1$ when z is a square and $s = -1$ when z is not a square. Many implementations simply compute $s \leftarrow z^{\frac{p-1}{2}}$. A successful fault injection in the computation $z \leftarrow x^3 + Ax^2 + x$, by skipping an instruction or changing the value randomly, ensures random input to `IsSquare` and so in about half of the cases the output will be flipped by $s \mapsto -s$. In the other half of the cases, the output of `IsSquare` remains s. The attacker knows the outcome of the non-faulty computation and can thus discard those outputs and continue with those where the orientation has been flipped successfully.

Remark 2. There are other ways to flip the orientation s. For example, one can also inject a random fault into x after s has been computed, which has a similar effect. The analysis and attack of Sections 4 and 5 apply to all possible ways to flip s, independent of the actual fault injection. The countermeasures introduced in Section 9 prevent all possible ways to flip s that we know of.

Faulting the Legendre symbol computation in `IsSquare`, in general, leads to a random $\mathbb{F}_p$-value as output instead of ± 1. The interpretation of this result is heavily dependent on the respective implementation. For instance, the CSIDH implementation from [15] interprets the output as boolean value by setting $s = 1$ if the result is $+1$, and -1 otherwise. In this case, faults mostly flip in one direction: from positive to negative orientation. Thus, faulting the computation of z is superior in our attack setting.

Elligator. Implementations using a 2-point strategy often use Elligator 2 [6]. On input of a random value, Elligator computes two points P and P' of opposite orientations. An `IsSquare` check is used to determine the orientation of P. If P has positive orientation, we set $P_+ \leftarrow P$ and $P_- \leftarrow P'$. Otherwise, set $P_+ \leftarrow P'$ and $P_- \leftarrow P$. Again, we can fault the input to this `IsSquare` check, which flips the assignments to P_+ and P_-; hence, the orientation of *both* points is flipped.

As before, this means that all isogenies computed using either of these points are pointing in the wrong direction. A notable exception is CTIDH, where two independent calls to Elligator are used to produce points for the 2-point strategy. This is due to security considerations, and the algorithmic and attack implications are detailed in Section 5.2.

4 Exploiting orientation flips

In this section, we analyze disorientation attacks in the context of generic 1-point and 2-point approaches for CSIDH (see Section 2.2).

A typical 1-point strategy implementation is given in Algorithm 1: first we sample a point P with orientation s, and then we determine a set S of indices (with the same orientation) for which we still need to compute isogenies.

Algorithm 1: Evaluation of CSIDH group action

```
Input: $A \in \mathbb{F}_p$ and a list of integers $(e_1, \ldots, e_n)$.
Output: $B \in \mathbb{F}_p$ such that $\prod [\mathfrak{l}_i]^{e_i} * E_A = E_B$
 1: while some $e_i \neq 0$ do
 2:     Sample a random $x \in \mathbb{F}_p$, defining a point $P$.
 3:     Set $s \leftarrow$ IsSquare$(x^3 + Ax^2 + x)$.
 4:     Let $S = \{i \mid e_i \neq 0,\ \mathrm{sign}(e_i) = s\}$. Restart with new $x$ if $S$ is empty.
 5:     Let $k \leftarrow \prod_{i \in S} \ell_i$ and compute $Q \leftarrow [\frac{p+1}{k}]P$.
 6:     for each $i \in S$ do
 7:         Set $k \leftarrow k/\ell_i$.
 8:         Compute $R \leftarrow [k]Q$. If $R = \infty$, skip this $i$.
 9:         Compute $\phi : E_A \to E_B$ with kernel $\langle R \rangle$.
10:         Set $A \leftarrow B$, $Q \leftarrow \phi(Q)$, and $e_i \leftarrow e_i - s$.
11: return $A$.
```

In Section 3, we defined an attack scenario that allows us to flip the orientation s in Line 3. If this happens, the net effect is that we will select an incorrect set S' with opposite orientation, and hence perform an isogeny walk in the *opposite* direction for all the indices in S'. Equivalently, the set S selected in Line 3 has opposite orientation to the point P. For simplicity, we will always fix the set S first and talk about the point P being flipped. We assume that we can successfully flip the orientation in any round r, and that we get the result of the faulty evaluation, which is some *faulty curve* $E_t \neq \mathfrak{a} * E$.

We first study the effect of orientation flips for full-order points in Section 4.2, and then discuss effects of torsion in Section 4.3 and Section 4.4. We organize the faulty curves into components according to their orientation and round in Section 4.5 and study the distance of components from different rounds in Section 4.6. In Section 4.7, we use faulty curves to recover the secret key $\mathfrak{a}$.

4.1 Implications of flipping the orientation of a point

In this section, all points will have full order, so Line 8 never skips an i.

Suppose we want to evaluate the group action $\prod_{i \in S} \mathfrak{l}_i * E_A$ for some set of steps S. Suppose we generate a negatively oriented point P, but flipped its orientation. This does not change the point (still negatively oriented), but if we use P to evaluate the steps in what we believe is the *positive* direction, we will in fact compute the steps in the negative direction: $E_f = \prod_{i \in S} \mathfrak{l}_i^{-1} * E_A$. More generally, if we want to take steps in direction s and use a point of opposite orientation, we actually compute the curve $E_f = \prod_{i \in S} \mathfrak{l}_i^{-s} * E_A$.

Suppose we flip the orientation of a point in one round of the isogeny computation $E_B = \mathfrak{a} * E_A$ and the rest of the computation is performed correctly. The resulting curve E_t is called a *faulty curve*. If the round was computing steps for isogenies in S with direction s, the resulting curve satisfies

$$E_B = \prod_{i \in S} \mathfrak{l}_i^{2s} * E_t,$$

that is, the faulty curve differs from the correct curve by an isogeny whose degree is given by the (squares of) primes ℓ_i for $i \in S$, the set S in the round we faulted. We call S the *missing set* of E_t.

Distance between curves. We define the *distance* d between two curves E and E' as the lowest number of different degrees for isogenies $\phi : E \to E'$. Note that the distance only tells us how many primes we need to connect two curves, without keeping track of the individual primes ℓ_i or their multiplicity. Specifically for a faulty curve with $E_B = \prod_{i \in S} \mathfrak{l}_i^{2s} * E_t$, we define the distance to E_B as the number of flipped steps $|S|$. Note that each $\mathfrak{l}_i$ appears as a square; this gets counted *once* in the distance.

Positive and negative primes. Suppose the secret key $\mathfrak{a}$ is given by the exponent vector (e_i). Then every ℓ_i is used to take e_i steps in direction $\operatorname{sign}(e_i)$. Define the set of *positive* primes $L_+ := \{i \mid e_i > 0\}$, *negative* primes $L_- := \{i \mid e_i < 0\}$, and neutral primes $L_0 := \{i \mid e_i = 0\}$.

For 1-point strategies and any faulty curve E_t with missing set S, we always have $S \subset L_+$ or $S \subset L_-$. However, using 2-point strategies, the sets S may contain positive and negative primes.

Example 1. Take CSIDH-512. Assume we flip the orientation $s \mapsto -s$ of the first point P. From Algorithm 1, we see the elements of S are exactly those i such that $|e_i| \geq 1$ and $\operatorname{sign}(e_i) = -s$. Therefore, we have $S = L_{-s}$.

4.2 Faulty curves and full-order points

We continue to assume that all points have full order, so Line 8 never skips an i, and analyze which faulty curves we obtain by flipping the orientation in round r. We treat the general case in Section 4.3 and Section 4.4.

Effective curves. For any strategy (cf. Section 2.2), the computation in round r depends on what happened in previous rounds. In a 2-point strategy, we sample both a negative and a positive point and use them to perform the isogenies in both directions. So assuming points of full order, the round-r computation and the set S do not depend on the previous round but only the secret key.

In a 1-point strategy, we sample 1 point per round, and only perform isogenies in the direction of that point. So the set S in round r depends on what was computed in previous rounds, not just the orientation of the sampled point. However, the computation in round r only depends on previous rounds with *the same orientation*, leading to the following definition.

Notation. Let $+$ and $-$ denote the positive and negative orientation, respectively. For a 1-point strategy, encode the choice of the orientation of the steps by a sequence of $\pm$. We denote the round r in which we flip the orientation by parentheses $(\cdot)$. We truncate the sequence at the moment of the fault because the rest of the computation is always computed correctly. Hence, $++(-)$ means any computation starting with the following three rounds: the first two rounds were positive, the third one should have been negative, but we flipped the orientation of the 3rd step and those negative steps were computed in the positive direction instead.

Consider a flip in the second round. There are four possible scenarios:

$+(+)$. Two positive rounds, but the second positive round was flipped and we took the steps in negative direction instead.
$+(-)$. One positive round, one negative round flipped to the positive direction.
$-(+)$. One negative round, one positive round flipped to the negative direction.
$-(-)$. Two negative rounds, the second one flipped to the positive direction.

All four cases are equally likely to appear for 1-point strategies, but result in different faulty curves. Since the computation only depends on previous rounds with the same orientation, the case $+(-)$ is easily seen to be the same as $(-)$ and $++(-)$. However, the cases $+(+)$ and $-(+)$ are different: the latter is equivalent to $(+)$. For example, in CSIDH, the set S for $(+)$ is $\{i \mid e_i \geq 1\}$, and the set S' for $+(+)$ is $\{i \mid e_i \geq 2\}$, differing exactly at the primes for which $e_i = 1$.

Effective round. Let $E^{r,+}$ be the faulty curve produced by the sequence $+\cdots+(+)$ of length r, and $E^{r,-}$ the curve produced by sequence $-\cdots-(-)$. We call the curves $E^{r,\pm}$ *effective round-r curves.*

Note that effective round-r curves can be produced from other sequences as well: $+(-)$ produces the effective round 1 curve $E^{1,-}$. Similarly, the sequence $++--+(-)$ is an effective round-3 sample $E^{3,-}$.

To get an effective round-r sample $E^{r,+}$ from a round n, the last sign in the sequence needs to be $(+)$, and the sequence contains a total of r pluses. We immediately get the following.

Lemma 4.1. *Assume we use a 1-point strategy. The probability to get an effective round-r sample if we successfully flip in round n is equal to $\binom{n-1}{r-1} \cdot \frac{1}{2^{n-1}}$.*

Remark 3. For a 2-point strategy, all curves resulting from a fault in round r are effective round-r curves.

Torsion sets $S^{r,+}$ and $S^{r,-}$. Define the set $S^{r,s}$ as the missing set of the effective round-r curve with orientation s, i.e., $E_B = \prod_{i \in S^{r,s}} \mathfrak{l}_i^{2s} * E^{r,s}$.

Example 2 (CSIDH). The sets $S^{1,\pm}$ were already discussed in Example 1. In general, $S^{r,+} = \{i \mid e_i \geq r\}$ and $S^{r,-} = \{i \mid e_i \leq -r\}$.

4.3 Missing torsion: faulty curves and points of non-full order

In Section 4.2, we worked under the unrealistic assumption that all points we encounter have full order. In this section, we relax this condition somewhat: we assume that every point had full order (and hence all isogenies were computed) up until round r, but the point P generated in round r potentially has smaller order. We call this the *missing torsion* case. The remaining relaxation of non-full order points in earlier rounds will be concluded in Section 4.4.

If the point P used to compute isogenies in round r does not have full order, the faulty curve E_t will differ from the effective round-r curve $E^{r,s}$ by the primes ℓ_i with $i \in S^{r,s}$ which are missing in the order of P.

Round-r faulty curves. For simplicity, assume that we are in round r, in the case $+\cdots+(+)$, and that none of the isogenies in the previous rounds failed. In round r, a negative point P is sampled, but we flip its orientation, so all the positive steps will be computed in the wrong direction.

If the point P has full order, we obtain the curve $E^{r,+}$ at the end of the computation, which differs from E_B exactly at primes contained in $S^{r,+}$. If, however, the point P does not have full order, a subset $S \subset S^{r,+}$ of steps will be computed, leading to a different faulty curve E_t. By construction, the curve E_t is related to E_B via $E_B = \prod_{i \in S} \mathfrak{l}_i^2 * E_t$.

Assume we repeat the fault T times, leading to different faulty curves E_t. Let $n(E_t)$ be the number of times the curve E_t occurs among the T samples.

For each such E_t, we know $E_B = \prod_{i \in S_t} \mathfrak{l}_i^{2s} * E_t$, where $S_t \subset S^{r,+}$ is determined by the order of P_t. As P_t is a randomly sampled point, it has probability $\frac{\ell_i - 1}{\ell_i}$ that its order is divisible by ℓ_i, and so probability $\frac{1}{\ell_i}$ that its order is not divisible by ℓ_i. This gives us directly the probability to end up at E_t: the order of the point P_t should be divisible by all ℓ_i for $i \in S_t$, but not by those ℓ_i for $i \in S^{r,+} \setminus S_t$. This is captured in the following result.

Proposition 4.2. *Let P_t be a random negative point, where we flip the orientation s to positive. The probability that we compute the faulty curve $E_t = \prod_{i \in S_t} \mathfrak{l}_i^{-2} * E_B$ is exactly $p_t = \prod_{i \in S_t} \frac{\ell_i - 1}{\ell_i} \cdot \prod_{i \in S^{r,+} \setminus S_t} \frac{1}{\ell_i}$.*

Proof. The probability of obtaining E_t is equal to the probability that the order of the point P_t is divisible by all the primes in S_t and not divisible by all the primes in $S^{r,+} \setminus S_t$. The first happens with probability $\prod_{i \in S_t} \frac{\ell_i - 1}{\ell_i}$; the second is an independent event happening with probability $\prod_{i \in S^{r,+} \setminus S_t} \frac{1}{\ell}$. □

Remark 4. Note that the CTIDH implementation artificially lowers the success probability of each point to match that of the smallest prime in the batch to hide which prime is handled. A result similar to Proposition 4.2 can be proven for fixed batches.

The expected number of appearances $n(E_t)$ of a curve E_t is $n(E_t) \approx p_t \cdot T$ for T runs. As $\frac{\ell_i - 1}{\ell_i} \geq \frac{1}{\ell_i}$ for all ℓ_i, the probability p_t is maximal when $S_t = S^{r,+}$. We denote this probability by $p^{r,+}$. Hence, the curve that is likely to appear the most in this scenario over enough samples, is the curve $E^{r,+}$ which we defined as precisely that curve with missing set $S^{r,+}$. For now, we focused solely on the positive curves. Taking into account the negative curves too, we get:

Corollary 4.3. *Let $E^{r,+} = \prod_{i \in S^{r,+}} \mathfrak{l}_i^{-2} * E_B$ and let $E^{r,-} = \prod_{i \in S^{r,-}} \mathfrak{l}_i^{2} * E_B$. Then $E^{r,+}$ and $E^{r,-}$ have the highest probability to appear among the effective round-r faulty curves. As a consequence, the largest two values $n(E)$ of all effective round-r curves are most likely $n(E^{r,+})$ and $n(E^{r,-})$*

Example 3 (CSIDH). Take the set $S^{1,+} = \{i \mid e_i \geq 1\}$ and let $p^{1,+}$ denote the probability that a random point P has order divisible by all primes in $S^{1,+}$. This probability depends on the secret key (e_i), but can be estimated if we collect enough faulty curves. Moreover, if $e_1 \neq 0$, then $\ell_1 = 3$ dominates either $p^{1,+}$ or $p^{1,-}$ through the relatively small probability of $2/3$ that P has order divisible by 3. Thus, if the largest pile of faulty curves is $E^{1,\pm}$, we expect $S^{1,\pm}$ not to contain 1. For instance, if e_1 is positive, $p^{1,-}$ is larger than $p^{1,+}$ and so we expect $n(E^{1,-})$ to be larger than $n(E^{1,+})$. In this case, we would expect to see another faulty curve E_t with $n(E_t)$ half the size of $n(E^{1,+})$; this curve E_t has *almost* full missing set $S^{1,+}$, but does not miss the 3-isogeny. That is, $S_t = S^{1,+} \setminus \{1\}$, with probability $p_t := \frac{1}{\ell_1} \cdot \frac{\ell_1}{\ell_1 - 1} \cdot p^{1,+} = \frac{1}{2} \cdot p^{1,+}$. This curve E_t is very "close" to $E^{1,+}$; they are distance 1 apart, precisely by $\mathfrak{l}_1^2$.

Remark 5. The precise probabilities $p^{r,+}$ and $p^{r,-}$ depend highly on the specific implementation. Given an implementation, the precise values of $p^{r,+}$ and $p^{r,-}$ allow for concrete estimates on the sizes of $n(E)$ for specific curves E.

Remark 6. Because primes that are missing in the order of P_t skip the misoriented steps, the curves in the neighborhood of $E^{r,+}$ differ by two ℓ_i-isogenies for $i \in S^{r,+} \setminus S_t$ in positive direction while those around $E^{r,-}$ differ by two ℓ_i-isogenies for $i \in S^{r,-} \setminus S_t$ in negative direction.

Distance between samples. We can generalize the above example for any two faulty curves E_t and $E_{t'}$ that are effective round-r samples of the same orientation, using Proposition 4.2. This describes which E_t are close to each other.

Corollary 4.4. *Let E_t and $E_{t'}$ both be effective round-r samples with the same orientation and missing torsion sets S_t and $S_{t'}$. Let S_Δ denote the difference in sets S_t and $S_{t'}$, i.e., $S_\Delta = (S_t \setminus S_{t'}) \cup (S_{t'} \setminus S_t)$. Then E_t and $E_{t'}$ are distance $|S_\Delta|$ apart, by*

$$E_t = \left(\prod_{i \in S_{t'} \setminus S_t} \mathfrak{l}_i^{2s} \cdot \prod_{i \in S_t \setminus S_{t'}} \mathfrak{l}_i^{-2s} \right) * E_{t'}.$$

In particular, any effective round-r curve E_t with orientation s is very close to $E^{r,s}$: since $S_t \subset S^{r,s}$, the difference S_Δ is small.

Example 4 (CSIDH). For a secret key $(2,3,1,2)$ in CSIDH with primes $L = \{3,5,7,11\}$, the first positive point with full torsion P will perform a 3, 5, 7 and 11-isogeny, so $S^{1,+} = \{1,2,3,4\}$ (with $S^{2,+} = \{1,2,4\}$ and $S^{3+} = \{2\}$). In one run, the first point P_{t_1} might only have $\{5,7,11\}$-torsion, while in another run the first point P_{t_2} might only have $\{3,7,11\}$-torsion. The faulty curves E_{t_1} and E_{t_2} differ from E^+ by two 3-isogenies and two 5-isogenies, respectively, and have a distance 2 towards each other: their S_Δ is $\{1,2\}$, so they are two $\{3,5\}$-isogenies apart. The two samples E_{t_1} and E_{t_2} therefore show that both 1 and 2 are in S^+ and show that $e_1 \geq 1$ and $e_2 \geq 1$.

Corollary 4.4 will be essential to recover information on $S^{r,+}$ out of the samples E_t, in a similar manner as the above example: Recovering small isogenies between samples allows us to deduce which i are in $S^{r,+}$ or $S^{r,-}$, and so leaks information about e_i.

4.4 Torsion noise

Orthogonally to Section 4.3, we now examine the case that missing torsion occurred in an earlier round than the round we are faulting.

Example 5 (CSIDH). Suppose that $e_1 = 1$ and that in the first positive round, the point generated in Line 2 of Algorithm 1 had order not divisible by ℓ_1, but all other points have full order. Thus, the ℓ_1-isogeny attempt fails in the first positive step. Consider now the second positive round. From Section 4.2, we would expect to be computing steps in $S^{2,+} = \{i \mid e_i \geq 2\}$. But no ℓ_1-isogeny has been computed in the first round, so it will be attempted in this second positive round. If we now fault the second round, we obtain a faulty curve that is *also missing* ℓ_1, that is, $E_t = \mathfrak{l}_1^{-2} * E^{2,+}$. Notice also that unlike the faulty curves from 4.3, the positively oriented isogeny goes from E_t *towards* $E^{2,+}$

Note also that in this scenario if $e_1 = 2$, a fault in round 2 would still result in the curve $E^{2,+}$, because the set $S^{2,+}$ contains ℓ_1 already, and so the missed ℓ_1-isogeny from round 1 will be computed in later rounds.

We refer to the phenomenon observed in Example 5 as *torsion noise*. More concretely, torsion noise happens when we fault the computation in round r for a run which is computing an ℓ_i-isogeny in round r for $|e_i| < r$ because it was skipped in a previous round.

Torsion noise is rarer than missing torsion but can still happen: the isogeny computation needs to fail and the fault must come when we are "catching up" with the computation. For CSIDH, torsion noise can only happen if $r > |e_i|$ and the computation of the ℓ_i-isogeny failed in at least $r - |e_i|$ rounds. Torsion noise is unlikely for large ℓ_i because the probability that an isogeny fails is about $1/\ell_i$.

For small primes, such as $\ell_i \in \{3, 5, 7\}$, we observe a lot of torsion noise. This can slightly affect the results as described in Section 4.3, but has no major impact on the results in general. Concretely, torsion noise may make it impossible to determine the correct e_i for the small primes given a small number of faulted curves. Nevertheless, their exact values can be brute-forced at the end of the attack.

Remark 7 (Orientation of torsion noise). Faulty curves affected by torsion noise require contrarily oriented isogenies to the curves $E^{r,s}$ than the remaining faulty curves. Therefore, if torsion noise happens and we find a path from such a curve $E_t \to E^{r,s}$, then we can infer not just the orientation of the primes in this path, but often also bound the corresponding exponents e_i.

4.5 Connecting curves from the same round

Suppose we have a set of (effective) round-r faulty curves with the same orientation s, and that r and s are fixed. In Corollary 4.4, we show that such curves are close to each other. In particular, the path from E_t to $E^{r,s}$ uses only degrees contained in the set $S^{r,s}$. Finding short paths among faulty curves gives us information about $S^{r,s}$, and hence about the secret key.

Component graphs. Starting from a set $\{E_t\}$ of round-r faulty curves with orientation s, we can use them to define the graph $G^{r,s}$ as follows: The vertices of $G^{r,s}$ are given by $\{E_t\}$, and the edges are steps between the curves, labeled by i if the curves are connected by two ℓ_i-isogenies.

For convenience, we sparsify the graph $G^{r,s}$ and regard it as a tree with the curve $E^{r,s}$ as the root.

Edges. Starting from a set of faulty curves, it is easy to build the graphs $G^{r,s}$. We can identify the *roots* of these graphs $E^{r,s}$ using Corollary 4.3. Then the distance from the root to any round-r faulty curve with the same orientation is small (cf. Corollary 4.4). Therefore, we can find the edges by applying short walks in the isogeny graph. Note that edges of $G^{r,s}$ give information on $S^{r,s}$.

Remark 8 (Missing vertices). If we do not have enough faulty curves $\{E_t\}$, it may not be possible to connect all the curves with single steps (understood as isogenies of square degree, see Corollary 4.4). For convenience, we assume that we have enough curves. In practice, we include in the graph $G^{r,s}$ any curve on the path between E_t to $E^{r,s}$ (again, taking steps with square prime degree).

Remark 9 (Components). We imagine the graphs $G^{r,s}$ as subgraphs of the *isogeny graph* of supersingular elliptic curves with edges given by isogenies. Computing short paths from $E^{r,s}$ will give us enough edges so that we can consider the graphs $G^{r,s}$ to be connected. Hence we call them *components*.

Secret information An effective round-r faulty curve E_t with torsion set $S_t \subset S^{r,+}$ can easily be connected by a path with labels $S^{r,+} \setminus S_t$. Moreover, the orientation $E^{r,+} \to E_t$ is positive. Therefore, we can identify which components are positive, and all the labels of the edges are necessarily in $S^{r,+}$, that is, the prime ℓ_i is positive.

Torsion noise can be recognized from the opposite direction of the edges (see Remark 7). For such an edge, the label $i \notin S^{r,+}$ but the prime ℓ_i is still positive.

In either case, the components $G^{r,s}$ give us the orientation of all the primes occurring as labels of the edges.

Sorting round-r samples. Suppose we are given a set of round-r faulty curves $\{E_t\}$, but we do not have information about the orientation yet. We can again use Corollary 4.3 to find the root of the graph; then we take small isogeny steps until we have two connected components G_1, G_2. It is easy to determine the direction of the edges given enough samples; ignoring torsion noise, the positively oriented root will have outgoing edges.

In summary, we try to move curves E_t from a pile of unconnected samples to one of the two graphs by finding collisions with one of the nodes in $G^{r,+}$ resp. $G^{r,-}$. The degrees of such edges reveal information on $S^{r,+}$ and $S^{r,-}$: An edge with label i in $G^{r,+}$ implies $i \in S^{r,+}$, and analogously for $G^{r,-}$ and $S^{r,-}$. Figure 1 summarizes the process, where, e.g., $E^{r,+} \to E_7$ shows missing torsion and $E_8 \to E^{r,+}$ is an example of torsion noise.

4.6 Connecting the components $G^{r,s}$

Now, we describe how we can connect the components $G^{r,s}$ for different rounds r. The distance of these components is related to the sets $S^{r,+}$ and $S^{r,-}$. We then show that it is computationally feasible to connect the components via a meet-in-the-middle attack. Connecting two components gives us significantly more knowledge on the sets $S^{r,+}$ and $S^{r,-}$, such that connecting all components is enough to reveal the secret $\mathfrak{a}$ in Section 4.7.

Information from two connected components. We start with an example;

Example 6 (CSIDH). Recall that we have $S^{r,+} = \{i \mid e_i \geq r\}$, and so $E^{r,+} = \prod_{i \in S^{r,+}} \mathfrak{l}_i^{-2} * E_B$. This means that, e.g., we have $S^{3,+} \subset S^{2,+}$, and $E^{2,+}$ has a

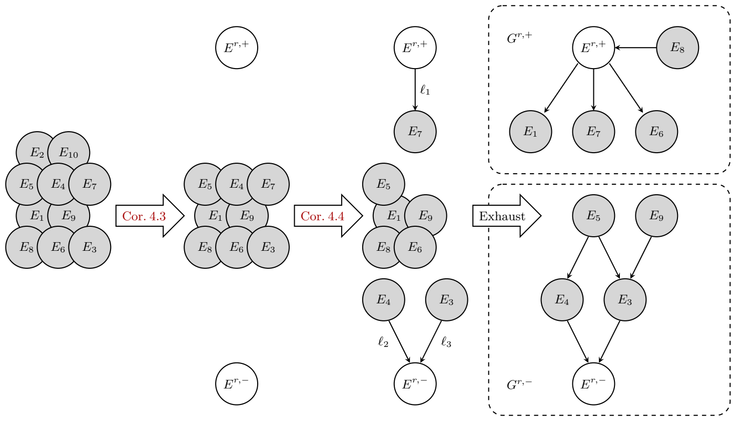

Fig. 1: Building up the component graphs of faulty curves.

larger distance from E_B than $E^{3,+}$. The path between $E^{3,+}$ and $E^{2,+}$ then only contains steps of degrees ℓ_i such that $i \in S^{2,+} \backslash S^{3,+}$, so $e_i = 2$. In general, it is easy to see that finding a single isogeny that connects a node E_{t_3} from $G^{3,+}$ and a node E_{t_2} in $G^{2,+}$ immediately gives the connection from $E^{3,+}$ to $E^{2,+}$. Hence, we learn all ℓ_i with $e_i = 2$ from the components $G^{3,+}$ and $G^{2,+}$.

In the general case, if we find an isogeny between two such graphs, say $G^{r,+}$ and $G^{r',+}$, then we can compute the isogeny between the two roots $E^{r,+}$ and $E^{r',+}$ of these graphs. The degree of this isogeny $E^{r,+} \to E^{r',+}$ describes precisely the *difference* between the sets $S^{r,+}$ and $S^{r',+}$. The example above is the special case $r' = r + 1$, and in CSIDH we always have $S^{(r+1),+} \subset S^{r,+}$, so that the difference between $S^{r,+}$ and $S^{(r+1),+}$ is the set of ℓ_i such that $e_i = r$. In other CSIDH-variants, such sets are not necessarily nested, but connecting the components still reveals e_i as Section 4.7 will show. In general, we connect two subgraphs by a distributed meet-in-the-middle search which finds the shortest connection first.

Distances between connected components. As we have shown, connecting two components $G^{r,+}$ and $G^{r',+}$ is equivalent to finding the difference in sets $S^{r,+}$ and $S^{r',+}$. The distance between these sets heavily depends on the implementation, as these sets are determined by the key $\mathfrak{a}$ and the evaluation of this key. For example, in CSIDH-512, the difference between $S^{r,+}$ and $S^{(r+1),+}$ are the $e_i = r$, which on average is of size $\frac{74}{11} \approx 6.7$. In practice, this distance roughly varies between 0 and 15. For an implementation such as CTIDH-512, the sets $S^{r,+}$ are smaller in general, on average of size 7, and the difference between such sets is small enough to admit a feasible meet-in-the-middle connection. See Section 6 for more details on how we connect these components in practice.

4.7 Revealing the private key

So far, we showed how connecting different components $G^{r,+}$ and $G^{r',+}$ reveals information on the difference between the sets $S^{r,+}$ and $S^{r',+}$. In this section, we show that assuming that all components are connected, we are able to derive the secret $\mathfrak{a}$. This wraps up Section 4: Starting with orientation flips in certain rounds r, we can derive the secret $\mathfrak{a}$ from the resulting graph structure, assuming enough samples.

From differences of sets to recoveries of keys. By connecting the graphs of all rounds, including the one-node-graph consisting of just the correct curve E_B, we learn the difference between the sets $S^{r,+}$ and $S^{(r+1),+}$ for all rounds r (as well as for $S^{r,-}$ and $S^{(r+1),-}$). A single isogeny from some $G^{r,+}$ to $E_B = \mathfrak{a} * E_A$ then recovers $S^{r,+}$ for this round r: Such an isogeny gives us an isogeny from $E^{r,+} = \prod_{i \in S^{r,+}} \mathfrak{l}_i^{-2} * E_B$ to E_B, whose degree shows us exactly those $\ell_i \in S^{r,+}$. From a connection between the components $G^{r,+}$ and $G^{r',+}$, we learn the difference in sets $S^{r,+}$ and $S^{r',+}$. From $S^{r,+}$, we can then deduce $S^{r',+}$.

Therefore, if all graphs $G^{r,+}$ for different r are connected, and we have at least one isogeny from a node to E_B, we learn the sets $S^{r,+}$ for all rounds r (and equivalently for $S^{r,-}$). From the knowledge of all sets $S^{r,+}$ and $S^{r,-}$ we then learn $\mathfrak{a} = (e_i)$: the sign of e_i follows from observing in which of the sets $S^{r,+}$ or $S^{r,-}$ the respective ℓ_i appears, and $|e_i|$ equals the number of times of these appearances.

In practice however, due to missing torsion and torsion noise, connecting all components may not give us the *correct* sets $S^{r,+}$ resp. $S^{r,-}$. In such a case, one can either gather more samples to gain more information, or try to brute-force the difference. In practice, we find that the actual set $S^{r,+}$ as derived from $\mathfrak{a}$ and the set $\tilde{S}^{r,+}$ derived from our attack (leading to some $\mathfrak{a}'$) always have a small distance. A simple meet-in-the-middle search between $\mathfrak{a}' * E_A$ and $\mathfrak{a} * E_A$ then quickly reveals the errors caused by missing torsion and torsion noise.

4.8 Complexity of recovering the secret $\mathfrak{a}$

The full approach of this section can be summarized as follows:

1. Gather enough effective round-r samples E_t per round r, using Lemma 4.1.
2. Build up the components $G^{r,+}$ and $G^{r,-}$ using Corollaries 4.3 and 4.4.
3. Connect components to learn the difference in sets $S^{r,+}$ and $S^{r',+}$.
4. Compute the sets $S^{r,+}$ and $S^{r,-}$ for every round and recover $\mathfrak{a}$.

The overall complexity depends on the number of samples per round, but is in general dominated by Step 3. For Step 2, nodes are in most cases relatively close to the root $E^{r,+}$ or to an already connected node E_t, as shown in Corollary 4.4.

For Step 3, components are usually further apart than nodes from Step 2. In general, the distance between components $G^{r,+}$ and $G^{r',+}$ depends heavily on the specific design choices of an implementation. In a usual meet-in-the-middle approach, where n is the number of ℓ_i over which we need to search and d is the distance between $G^{r,+}$ and $G^{r',+}$, the complexity of finding a connection is $\mathcal{O}(\binom{n}{d/2})$. Note that we can use previous knowledge from building components or finding small-distance connections between other components to reduce the search space and thus minimize n for subsequent connections. We analyze this in detail for specific implementations in Section 5.

5 Case studies: CSIDH and CTIDH

We previously defined a general strategy in four steps. In practice, those steps are dependent on the actual implementation. Concretely, we select two main implementations: CSIDH-512 and CTIDH-512. We discuss CSIDH-512 in Section 5.1, CTIDH-512 in Section 5.2, and we analyze other implementations in Section 5.3.

In this section we will specialize to inputting E_0 into the target which thus computes a faulty version of $E_B = \mathfrak{a} * E_0$, its own public key.

5.1 Breaking CSIDH-512

The primes used in CSIDH-512 [15] are $L = \{3, 5, \ldots, 377, 587\}$, and exponent vectors are sampled as $(e_i) \in \{-5, \ldots, 5\}^{74}$ uniformly at random. For any $k \in \{-5, \ldots, 5\}$ we expect about $\frac{1}{11} \cdot 74$ primes ℓ_i with $e_i = k$; this count obeys a binomial distribution with parameters $(74, 1/11)$. In particular, we expect to see about $\frac{5}{11} \cdot 74 \approx 33.6$ positive and negative primes each, and about $\frac{1}{11} \cdot 74 \approx 6.7$ neutral primes.

In CSIDH-512, the group action is evaluated as displayed in Algorithm 1, using a 1-point strategy. In particular, after generating a point with orientation s, we set $S = \{i \mid e_i \neq 0,\ \mathrm{sign}(e_i) = s\}$. If the value of s is flipped, we set $S = \{i \mid e_i \neq 0,\ \mathrm{sign}(e_i) = -s\}$, but we perform the steps in direction s. The secret-key recovery follows the four steps defined in Section 4.8, with the following specifications.

Building components $G^{r,+}$ and $G^{r,-}$. Step 2 of the attack on CSIDH-512 works exactly as described in Section 4.5. If E_t and $E_{t'}$ are effective samples from the same round with the same orientation, their distance is small (Corollary 4.4). We can thus perform a neighborhood search on all of the sampled curves until we have 10 connected components $G^{r,\pm}$ for $r \in \{1, \ldots, 5\}$, as in Figure 1. This step is almost effortless: most curves will be distance 1 or 2 away from the root $E^{r,s}$. In practice, using round information and number of occurrences, we identify the 10 curves $E^{r,\pm}$ for $r = 1, \ldots, 5$, and explore all paths of small length from those 10 curves, or connect them via a meet-in-the-middle approach (e.g., using `pubcrawl`, see Section 6).

The degrees of the isogenies corresponding to the new edges in $G^{r,\pm}$ reveal information on the sets $S^{r,\pm}$, which can be used to reduce the search space when connecting the components $G^{r,\pm}$.

Filter-and-break it, until you make it. Step 3 is the most computationally intensive step, as it connects 11 components ($G^{r,\pm}$ and E_B) into a single large connected component. We argue that it is practical for CSIDH-512.

More specifically, we want to find connections between $G^{r,\pm}$ and $G^{(r+1),\pm}$, as well as connections from $G^{5,\pm}$ to E_B. This gives us 10 connections, corresponding to the gaps $\{i \mid e_i = k\}$ for $k \in [-5, 5] \setminus \{0\}$. Figure 2 shows an abstraction of this large connected component. Since there are 74 primes in total, and only 10

$$\begin{tikzcd}
G^{1,+} \arrow[r, "S^{1,+} \setminus S^{2,+}"] & G^{2,+} \arrow[r, "S^{2,+} \setminus S^{3,+}"] & G^{3,+} \arrow[r, "S^{3,+} \setminus S^{4,+}"] & G^{4,+} \arrow[r, "S^{4,+} \setminus S^{5,+}"] & G^{5,+} \arrow[dr, "S^{5,+}"] & \\
 & & & & & E_B \arrow[dl, "S^{5,-}"] \\
G^{1,-} & G^{2,-} \arrow[l, "S^{1,-} \setminus S^{2,-}"'] & G^{3,-} \arrow[l, "S^{2,-} \setminus S^{3,-}"'] & G^{4,-} \arrow[l, "S^{3,-} \setminus S^{4,-}"'] & G^{5,-} \arrow[l, "S^{4,-} \setminus S^{5,-}"'] &
\end{tikzcd}$$

Fig. 2: Large connected component associated to an attack on CSIDH-512.

gaps, at least one of these gaps is at most 7 primes. If we assume that at least 5 of the exponents are 0 (we expect ≈ 7 to be 0), then the smallest distance is at most 6 steps. Such gaps are easily found using a meet-in-the-middle search, see Section 6.

Let us call *support* the set of isogeny degrees used in a meet-in-the-middle neighborhood search. We can certainly connect all the components by a naive meet-in-the-middle search with support $\{\ell_1, \dots, \ell_{74}\}$. However, for larger distances, trying out all possible isogenies is infeasible.

We search for the longer paths by adaptively changing the support. We start by finding short connections, and then use the information we learn from those to pick a smaller support for searching between certain components, i.e., *filter* some of the ℓ_i out of the support. We describe the procedure below.

First, we learn which components are positive and which are negative by identifying the components $G^{1,\pm}$ and considering the direction of the edges. Since effective round-1 samples do not have torsion noise, the root $E^{1,+}$ has only outgoing edges, whereas the root $E^{1,-}$ has only incoming edges. The labels of the edges of $G^{1,+}$ are necessarily positive primes, and all components with a matching label are necessarily positive, and the edges in that component are again positive primes. The same reasoning follows for negative components. So orienting the components is typically easy, given enough samples.

Next, all the i that appear as degrees of edges in $G^{r,+}$ for any r are necessarily positive, similarly, all primes appearing in edges in $G^{r,-}$ are necessarily negative. But positively oriented components can only be connected by positive primes, so when searching for paths, we can remove from the support all the primes that we know are negative.

After finding the first connection we restrict the support even more: we know that any i appears in at most *one* connection. Hence, whenever we find a connection, we get more information about orientation and can reduce the support for further searches, allowing us to find larger connections. We repeat this procedure with more and more restrictions on the support until we find the full connected component.

Recovering the secret key. From the connected components, we recover all of the sets $S^{r,\pm}$ and we compute the secret key as described in Section 4.7.

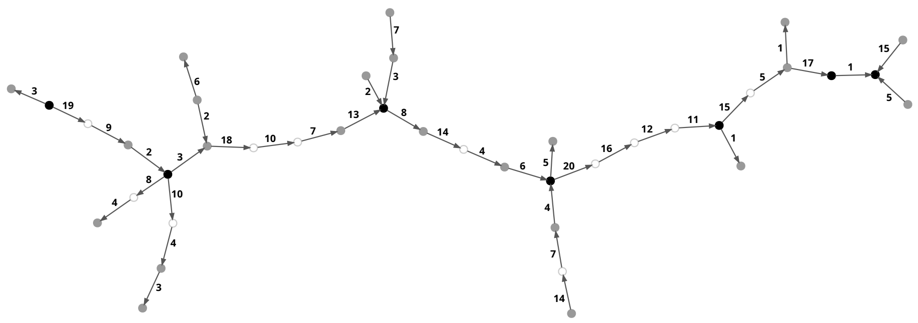

Fig. 3: Example isogeny graph of faulty curves obtained from attacking the fictitious CSIDH-103 implementation from Example 7. An edge labeled i denotes the isogeny step $\mathfrak{l}_i$. The E_B curve and the root faulty curves $E^{r,s}$ are rendered in black (from left to right: $E^{1,+}$, $E^{2,+}$, $E^{3,+}$, E_B, $E^{3,-}$, $E^{2,-}$, $E^{1,-}$), other faulty curves appearing in the dataset are gray, and white circles are "intermediate" curves discovered while connecting the components.

Example 7 (Toy CSIDH-103). Figure 3 shows the resulting connected graph for a toy version of CSIDH using Algorithm 1 with the first $n = 21$ odd primes and private keys in $\{-3, \ldots, +3\}^n$. Each round was faulted 10 times.

The distances between the components are very small and hence connecting paths are readily found. We sparsify the graph to plot it as a spanning tree; the edges correspond to positive steps of the degree indicated by the label. This graph comes from the secret key

$$(-1, +1, +2, +3, -2, +3, +2, +3, +1, +2, -3, -3, +2, +3, -2, -3, -2, +2, +1, -3,\ 0).$$

Required number of samples. Recovering the full secret exponent vector in CSIDH-512 equates to computing the sets $S^{r,+}$ and $S^{r,-}$ for $r \in \{1, \ldots, 5\}$. Recall that to compute these sets we need to build a connected component including subcomponents $G^{r,+}$ and $G^{r,-}$ for $r \in \{1, \ldots, 5\}$, and E_B (the one-node-graph consisting of just the public key). We build the components $G^{r,+}$ and $G^{r,-}$ by acquiring enough effective round-r samples. More effective round-r samples may give more vertices in $G^{r,\pm}$, and more information about $S^{r,\pm}$.

We denote the number of successful fault injections in round r by T_r and the total number of samples by $T = \sum T_r$. A first approach is to inject in round r until the probability is high enough that we have enough effective round-r samples. For CSIDH-512, we take $T_1 = 16$, $T_2 = 16$, $T_3 = 32$, $T_4 = 64$ and $T_5 = 128$, so that $T = 256$. From Lemma 4.1, we then expect 8 round-5 samples (4 per orientation) and the probability that we do not get any of the elements of $G^{5,+}$ or $G^{5,-}$ is about 1.7%.

This strategy can be improved upon. Notice that we need round-5 samples, and so in any case we need T_5 rather large (in comparison to T_i with $i < 5$) to ensure we get such samples. But gathering samples from round 5 already gives us many samples from rounds before. Using Lemma 4.1 with $T_5 = 128$, we get on average 8 effective round-1 samples, 32 effective round-2 samples, 48 effective round-3 samples, 32 effective round-4 samples and 8 effective round-5 samples. In general, attacking different rounds offers different tradeoffs: attacking round 9 maximizes getting effective round-5 samples, but getting a round-1 sample in round 9 is unlikely. Faulting round 1 has the benefits that all faulty curves are effective round-1 curves, making them easy to detect in later rounds; that no torsion noise appears; and that missing torsion quickly allows to determine the orientation of the small primes, reducing the search space for connecting the components.

5.2 Breaking CTIDH-512

CTIDH [3] partitions the set of primes ℓ_j into b batches, and bounds the number of isogenies per batch. For a list $N \in \mathbb{Z}^b_{>0}$ with $\sum N_k = n$ and a list of non-negative bounds $m \in \mathbb{Z}^b_{\geq 0}$ define the keyspace as

$$\mathcal{K}_{N,m} := \left\{ (e_1, \ldots, e_n) \in \mathbb{Z}^n \;\middle|\; \textstyle\sum_{j=1}^{N_i} |e_{i,j}| \leq m_i \text{ for } 1 \leq i \leq b \right\},$$

where $(e_{i,j})$ is a reindexed view of (e_i) given by the partition into batches.

CTIDH-512 uses 14 batches with bounds $m_i \leq 18$, requiring at least 18 rounds. In every round, we compute one isogeny per batch; using a 2-point strategy, we compute isogenies in both positive and negative direction. So, all round-r samples are effective round-r samples.

Injecting faults. To sample oriented points, CTIDH uses the Elligator-2 map twice. First, Elligator is used to sample two points P_+ and P_- on the starting curve E_A. A direction s is picked to compute an isogeny, the point P_s is used to take a step in that direction to a curve $E_{A'}$, and the point P_s is mapped through the isogeny. Then another point P'_{-s} is sampled on $E_{A'}$ using Elligator.

We will always assume that we inject a fault into only one of these two Elligator calls (as in Section 3). Hence, as for CSIDH and 1-point strategies, we again always obtain either positively or negatively oriented samples.

Different rounds for CTIDH-512. Per round, CTIDH performs one $\ell_{i,j}$ per batch $\mathcal{B}_i$. Within a batch, the primes $\ell_{i,j}$ are ordered in ascending order: if the first batch is $\mathcal{B}_1 = \{3, 5\}$ and the exponents are $(2, -4)$, then we first compute 2 rounds of 3-isogenies in the positive direction, followed by 4 rounds of 5-isogenies in the negative direction. We can visualize this as a queue $[3+, 3+, 5-, 5-, 5-, 5-]$ (padded on the right with dummy isogenies for the remaining rounds up to m_1). CTIDH inflates the failure of each isogeny to that of the smallest prime in the batch to hide how often each prime is used; in our example, the failure probability is 1/3.

This implies that the sets $S^{r\pm}$ contain precisely the r-th prime in the queue for the batch $\mathcal{B}_i$. With 14 batches and an equal chance for either orientation, we expect that each $S^{r\pm}$ will contain about 7 primes. Furthermore, each set $S^{r\pm}$ can contain only one prime per batch $\mathcal{B}_i$.

The small number of batches and the ordering of primes within the batches make CTIDH especially easy to break using our disorientation attack.

Components for CTIDH-512. Given enough samples, we construct the graphs $G^{r,s}$; the slightly higher failure probability of each isogeny (because of inflating) somewhat increases the chances of missing torsion and torsion noise.

The distance of the root curves $E^{r,s}$ to the non-faulted curve E_B is bounded by the number of batches. Per round r, the sum of the distances of $E^{r,\pm}$ to E_B is at most 14, so we expect the distance to be about 7.

The distance between two graphs $G^{r,s}$ and $G^{(r+1),s}$ is often much smaller. We focus on positive orientation (the negative case is analogous). The distance between $G^{r,+}$ to $G^{(r+1),+}$ is given by the set difference of $S^{r,+}$ and $S^{(r+1),+}$. If these sets are disjoint and all primes in round r and $r+1$ are positive, the distance is 28, but we expect significant overlap: The set difference contains the indices i such that either the last ℓ_i-isogeny is computed in round r or the first ℓ_i-isogeny is computed in round $r+1$. Note that these replacements need not come in pairs. In the first case, the prime ℓ_i is replaced by the next isogeny ℓ_j from the same batch only if ℓ_j is also positive. In the second case, the prime ℓ_i might have followed a negative prime that preceded it in the batch.

Therefore, given $S^{r,+}$, one can very quickly determine $S^{(r+1),+}$ by leaving out some ℓ_i's or including subsequent primes from the same batch. In practice, this step is very easy. Finding one connection $E_B \to E^{r,+}$ determines some set $S^{r,+}$, which can be used to quickly find other sets $S^{r',+}$. This approach naturally also works going backwards, to the set $S^{(r-1),+}$.

Directed meet-in-the-middle. Using a meet-in-the-middle approach, we compute the neighborhood of E_B and all the roots $E^{r,\pm}$ (or components $G^{r,\pm}$) of distance 4. This connects E_B to all the curves at distance at most 8. Disregarding orientation and information on batches, if we have N curves that we want to connect, the naive search will require about $2 \cdot \binom{74}{4} \cdot N \approx 2^{21} \cdot N$ isogenies. The actual search space is even smaller as we can exclude all paths requiring two isogenies from the same batch.

Moreover, isogenies in batches are in ascending order. So, if in round r we see that the 3rd prime from batch $\mathcal{B}_i$ was used, none of the rounds $r' > r$ involves the first two prime, and none of the rounds $r' < r$ can use the fourth and later primes from the batch for that direction.

Late rounds typically contain many dummy isogenies and the corresponding faulty curves are especially close to the public key. We expect to rapidly recover $S^{r,\pm}$ for the late round curves, and work backwards to handle earlier rounds.

Required number of samples. In CTIDH, we can choose to inject a fault into the first call of Elligator or the second one. We do not see a clear benefit of prioritizing either call. Unlike for CSIDH and 1-point strategies, there is no clear benefit from targeting a specific round.

Assume we perform c successful faults per round per Elligator call, expecting to get samples for both orientations per round. As CTIDH-512 performs 18 rounds (in practice typically up to 22 because of isogeny steps failing), we require $T = 18 \cdot 2 \cdot c$ successful flips. It seems possible to take $c = 1$ and hence $T = 36$ (or up to $T = 44$) samples.

With just one sample per round r (and per orientation s), the torsion effects will be significant and we will often not be able to recover $S^{r,s}$ precisely. Let $\tilde{S}^{r,s}$ denote the index set recovered for round r and sign s. We can correct for some of these errors, looking at $\tilde{S}^{r',\pm}$ for rounds r' close to r. Consider only primes from the same batch $\mathcal{B}$, then the following can happen:

- *No* prime from $\mathcal{B}$ is contained in either $\tilde{S}^{r,+}$ or $\tilde{S}^{r,-}$: all primes from $\mathcal{B}$ are done or *missing torsion* must have happened. We can examine the primes from the batch $\mathcal{B}$ which occur in neighboring rounds $\tilde{S}^{(r\pm1),\pm}$ and use the ordering in the batch to obtain guesses on which steps should have been computed if any.
- *One* prime from $\mathcal{B}$ is contained in $\tilde{S}^{r,+} \cup \tilde{S}^{r,-}$: we fix no errors.
- *Two* primes from $\mathcal{B}$ are contained in $\tilde{S}^{r,+} \cup \tilde{S}^{r,-}$: the smaller one must have come from torsion noise in a previous round and can be removed.

Remark 10. It is possible to skip certain rounds to reduce the number of samples, and recover the missing sets $S^{r,s}$ using information from the neighboring rounds. We did not perform the analysis as to which rounds can be skipped, we feel that already two successful faults per round are low enough.

Even a partial attack (obtaining information only from a few rounds) reveals a lot about the secret key thanks to the batches being ordered, and can reduce the search space for the secret key significantly. One may also select the rounds to attack adaptively, based on the information recovered from $S^{r,s}$.

Recovering the secret key. Once we recover all the sets $S^{r,s}$, the secret key can be found as $\mathfrak{a} = \prod_r \left(\prod_{i \in S^{r,+}} \mathfrak{l}_i \cdot \prod_{j \in S^{r,-}} \mathfrak{l}_j^{-1} \right)$. Note that as before, if we misidentify $S^{r,s}$ due to torsion effects, we may have to perform a small search to correct for the mistakes.

5.3 Other variants of CSIDH

In this section, we discuss some of the other implementations of CSIDH: all of these use `IsSquare` checks in the process of point sampling and are vulnerable to our attack. We analyze SIMBA [30], dummy-free implementations [1,16,18], and SQALE [17].

SIMBA. Implementations using SIMBA [30] can be attacked similarly to CSIDH (cf. Section 5.1). SIMBA divides the n primes ℓ_i into m *prides* (batches), and each round only computes ℓ_i-isogenies from the same pride. That is, each round only involves up to $\lceil n/m \rceil$ isogenies, and the setup of the prides is publicly known.

In each round, fewer isogenies are computed, the sets $S^{r,s}$ are smaller and the distances between the components $G^{r,s}$ are shorter. It is therefore easier to find isogenies connecting the components, and recover the secret key.

Dummy-free CSIDH. Dummy-free implementations [1,16,18] replace pairs of dummy ℓ_i-isogenies by pairs of isogenies that effectively cancel each other [16]. This is due to the fact that $\mathfrak{l}_i * (\mathfrak{l}_i^{-1} * E) = \mathfrak{l}_i^{-1} * (\mathfrak{l}_i * E) = E$. Thus, computing one ℓ_i-isogeny in positive direction and one ℓ_i-isogeny in negative direction has the same effect as computing two dummy ℓ_i-isogenies. However, this approach requires fixing the parity of each entry of the private key e_i, e.g., by sampling only even numbers from $[-10, 10]$ to reach the same key space size as before. The implementation of [16] therefore suffers a slowdown of factor 2. Nevertheless, such dummy-free implementations mitigate certain fault attacks, such as skipping isogenies, which in a dummy-based implementation would directly reveal if the skipped isogeny was a dummy computation and give respective information on the private key.

Dummy-free CSIDH [1] computes $|e_i|$ ℓ_i-isogenies per i in the appropriate direction, and then computes equally many ℓ_i isogenies in both directions which cancel out, until all required isogenies have been computed. For instance, for an even e_i sampled from $[-10, 10]$, choosing $e_i = 4$ would be performed by applying $\mathfrak{l}_i^1$ in the first 5 rounds, applying $\mathfrak{l}_i^{-1}$ in round 6 and 7, applying $\mathfrak{l}_i^1$ again in round 8 and 9, and finishing with $\mathfrak{l}_i^{-1}$ in round 10.

Notice that all isogenies start in the correct direction, and that we learn $|e_i|$ from disorientation faults if we know in which round the first $\mathfrak{l}_i$ is applied in the opposite direction. Therefore, if we apply the attack of Section 4 and learn all sets $S^{r,+}$ and $S^{r,-}$, we can determine e_i precisely. Even better, it suffices to only attack every second round: It is clear that each prime will have the same orientation in the third round as in the second round, in the fifth and fourth, et cetera. Due to the bounds used in [1], large degree ℓ_i do not show up in later rounds, which decreases the meet-in-the-middle complexity of connecting the components $G^{r,+}$ and $G^{(r+1),+}$ for later rounds r.

SQALE. SQALE [17] only uses exponent bounds $e_i \in \{-1, 1\}$. To get a large enough key space, more primes ℓ_i are needed; the smallest instance uses 221 ℓ_i. SQALE uses a 2-point strategy and only requires one round (keeping in mind the isogeny computation may fail and require further rounds).

Set $S^+ = S^{1,+} = \{i \mid e_i = 1\}$ and $S^- = S^{1,-} = \{i \mid e_i = -1\}$. If the sampled points in round 1 have full order, the round 1 faulty curves are either:

- the 'twist' of E_B: all the directions will be flipped (if both points are flipped),
- or the curve $E^+ = (\prod_{S^+} \mathfrak{l}_i^{-2}) * E_B$, if the positive point was flipped,
- or the curve $E^- = (\prod_{S^-} \mathfrak{l}_i^{2}) * E_B$, if the negative point was flipped.

As $|S^+| \approx |S^-| \approx n/2 > 110$, we will not be able to find an isogeny to either of these curves using a brute-force or a meet-in-the-middle approach.

However, SQALE samples points randomly, and some of the isogeny computation will fail, producing faulty curves close to $E^\pm$ (and curves with the same orientation will be close to each other, as in Section 4.5). Getting enough faulty curves allows the attacker to get the orientation of all the primes ℓ_i, and the orientation of the primes is exactly the secret key in SQALE. We note that [18] in another context proposes to include points of full order into the system parameters and public keys such that missing torsion and torsion noise do not occur. If this is used for SQALE, our attack would not apply.

6 The `pubcrawl` tool

The post-processing stage of our attack relies on the ability to reconstruct the graph of connecting isogenies between the faulty CSIDH outputs. We solve this problem by a meet-in-the-middle neighbourhood search in the isogeny graph, which is sufficiently practical for the cases we considered. In this section, we report on implementation details and performance results for our `pubcrawl` software.[7]

We emphasize that the software is *not* overly specialized to the fault-attack setting and may therefore prove useful for other "small" CSIDH isogeny searches appearing in unrelated contexts.

Algorithm. `pubcrawl` implements a straightforward meet-in-the-middle graph search: Grow isogeny trees from each input node simultaneously and check for collisions; repeat until there is only one connected component left. The set of admissible isogeny degrees ("support") is configurable, as are the directions of the isogeny steps ("sign", cf. CSIDH exponent vectors), the maximum number of isogeny steps to take from each target curve before giving up ("distance"), and the number of prime-degree isogenies done per graph-search step ("multiplicity", to allow for restricting the search to square-degree isogenies).

[7] The name refers to *crawl*ing the graph of *pub*lic keys, and a tour taking in several pubs or drinking places, with one or more drinks at each.

Size of search space. The number of vectors in $\mathbb{Z}^n$ of 1-norm $\leq m$ is [20, § 3]

$$G_n(m) = \sum_{k=0}^{m} \binom{n}{k} \binom{m-k+n}{n} .$$

Similarly, the number of vectors in $\mathbb{Z}^n_{\geq 0}$ of 1-norm $\leq m$ equals

$$H_n(m) = \sum_{k=0}^{m} \binom{k+n-1}{n-1} .$$

Implementation. The tool is written in C++ using modern standard library features, most importantly hashmaps and threading. It incorporates the latest version of the original CSIDH software as a library to provide the low-level isogeny computations. Public-key validation is skipped to save time. The shared data structures (work queue and lookup table) are protected by a simple mutex; more advanced techniques were not necessary in our experiments.

We refrain from providing detailed benchmark results for the simple reason that the overwhelming majority of the cost comes from computing isogeny steps in a breadth-first manner, which parallelizes perfectly. Hence, both time and memory consumption scale almost exactly linearly with the number of nodes visited by the algorithm.

Concretely, on a server with two Intel Xeon Gold 6136 processors (offering a total of 24 hyperthreaded Skylake cores) using GCC 11.2.0, we found that each isogeny step took between 0.6 and 0.8 core milliseconds, depending on the degree. Memory consumption grew at a rate of ≈ 250 bytes per node visited, although this quantity depends on data structure internals and can vary significantly. Example estimates based on these observations are given in Table 1.

There is no doubt that `pubcrawl` could be sped up if desired, for instance by computing various outgoing isogeny steps at once instead of calling the CSIDH library as a black box for each individually.

Code. The `pubcrawl` software is available at `https://yx7.cc/code/pubcrawl/pubcrawl-latest.tar.xz`.

7 Hashed version

As briefly mentioned in Remark 1, the attacker-observable output in Diffie–Hellman-style key agreements is not the shared elliptic curve, but a certain derived value. Typically, the shared elliptic curve is used to compute a key k using a key derivation function, which is further used for symmetric key cryptography. So we cannot expect to obtain (the Montgomery coefficient of) a faulty curve E_t but only a derived value such as $k = \text{SHA-256}(E_t)$ or $\text{MAC}_k(\texttt{str})$ for some known fixed string `str`.

The attack strategies from Section 4 and Section 5 exploit the connections between the various faulty curves. In this section, we argue that our attack extends

Table 1: Example cost estimates per target curve for various `pubcrawl` instances, assuming each isogeny step takes 0.7 milliseconds and consumes 250 bytes. For example, an isogeny walk of length up to 10 between two given curves can be recovered using approximately 10 core days and 300 gigabytes of RAM.

sign	|support|	distance	cardinality of search space	core time	memory
both	74	≤ 2	$11{,}101 \approx 2^{13.44}$	7.8 s	2.8 MB
both	74	≤ 3	$551{,}449 \approx 2^{19.07}$	6.4 min	137.9 MB
both	74	≤ 4	$20{,}549{,}801 \approx 2^{24.29}$	4.0 h	5.1 GB
both	74	≤ 5	$612{,}825{,}229 \approx 2^{29.19}$	5.0 d	153.2 GB
both	74	≤ 6	$15{,}235{,}618{,}021 \approx 2^{33.83}$	123.4 d	3.8 TB
both	74	≤ 7	$324{,}826{,}290{,}929 \approx 2^{38.24}$	7.2 y	81.2 TB
both	74	≤ 8	$6{,}063{,}220{,}834{,}321 \approx 2^{42.46}$	134.6 y	1.5 PB
both	74	≤ 9	$100{,}668{,}723{,}849{,}029 \approx 2^{46.52}$	2234.5 y	25.2 PB
one	74	≤ 2	$2{,}850 \approx 2^{11.48}$	2.0 s	712.5 kB
one	74	≤ 3	$73{,}150 \approx 2^{16.16}$	51.2 s	18.3 MB
one	74	≤ 4	$1{,}426{,}425 \approx 2^{20.44}$	16.6 min	356.6 MB
one	74	≤ 5	$22{,}537{,}515 \approx 2^{24.43}$	4.4 h	5.6 GB
one	74	≤ 6	$300{,}500{,}200 \approx 2^{28.16}$	2.4 d	75.1 GB
one	74	≤ 7	$3{,}477{,}216{,}600 \approx 2^{31.70}$	28.2 d	869.3 GB
one	74	≤ 8	$35{,}641{,}470{,}150 \approx 2^{35.05}$	288.8 d	8.9 TB
one	74	≤ 9	$328{,}693{,}558{,}050 \approx 2^{38.26}$	7.3 y	82.2 TB
both	37	≤ 2	$2{,}813 \approx 2^{11.46}$	2.0 s	703.2 kB
both	37	≤ 3	$70{,}375 \approx 2^{16.10}$	49.3 s	17.6 MB
both	37	≤ 4	$1{,}321{,}641 \approx 2^{20.33}$	15.4 min	330.4 MB
both	37	≤ 5	$19{,}880{,}915 \approx 2^{24.24}$	3.9 h	5.0 GB
both	37	≤ 6	$249{,}612{,}805 \approx 2^{27.90}$	2.0 d	62.4 GB
both	37	≤ 7	$2{,}691{,}463{,}695 \approx 2^{31.33}$	21.8 d	672.9 GB
both	37	≤ 8	$25{,}450{,}883{,}345 \approx 2^{34.57}$	206.2 d	6.4 TB
both	37	≤ 9	$214{,}483{,}106{,}715 \approx 2^{37.64}$	4.8 y	53.6 TB
one	37	≤ 3	$9{,}880 \approx 2^{13.27}$	6.9 s	2.5 MB
one	37	≤ 4	$101{,}270 \approx 2^{16.63}$	1.2 min	25.3 MB
one	37	≤ 5	$850{,}668 \approx 2^{19.70}$	9.9 min	212.7 MB
one	37	≤ 6	$6{,}096{,}454 \approx 2^{22.54}$	1.2 h	1.5 GB
one	37	≤ 7	$38{,}320{,}568 \approx 2^{25.19}$	7.5 h	9.6 GB
one	37	≤ 8	$215{,}553{,}195 \approx 2^{27.68}$	1.7 d	53.9 GB
one	37	≤ 9	$1{,}101{,}716{,}330 \approx 2^{30.04}$	8.9 d	275.4 GB

to this more realistic setting as long as the observable value is deterministically computable from E_t and collisions do not occur.

For simplicity, we will refer to the observable values as *hashes* of the faulty curves. Starting from a faulty curve E_t, we assume we can easily compute the hashed value $H(E_t)$, but we cannot recover E_t from the hash $h = H(E_t)$.

Recall the general strategy from Section 4: generate faulty curves, figure out the orientation of the primes from the torsion phenomena, generate neighborhood graphs, and find connecting paths between these graphs. Knowing the orientation of some primes helped reduce the possible degrees of the isogenies when applying `pubcrawl`, thus making the neighborhood search more efficient.

If we only see hashes of the faulty curves, we will not be able to easily form the neighborhood graphs and determine orientation. But from the frequency analysis (Corollary 4.3), we can identify the two most frequent new hashes h_1, h_2 per round as the probable hashes of $H(E^{r,\pm})$.

Example 8 (CSIDH). When faulting in the first round, the two most common hashed values are our best guesses for the hashes of $E^{1,\pm}$. Considering faults in round 2, we guess $H(E^{2,\pm})$ to be the most common hashes that have not appeared in round 1. Similarly for later rounds.

To identify the curves from the hashes, we run (one-sided) `pubcrawl` starting from E_B and hash all the curves found. We run `pubcrawl` with one orientation (or both, in parallel) until we recognize $H(E^{r,\pm})$ among the hashes. Then we have identified a curve $E^{r,s}$, and can run a small neighborhood search around $E^{r,s}$. We always hash all the curves and check whether any occurred as hashes of faulty curves, possibly gaining orientation information on some primes, which will make the next `pubcrawl` steps more efficient.

Most importantly: we can only do one-sided searches starting from a known curve to the hashes of faulty curves. This is in contrast to doing meet-in-the-middle attacks also starting from the faulty curves as in the previous sections. In particular, the only known curve at the beginning of the attack is E_B.

Example 9 (CSIDH-512). The distance of the curves $E^{r,s}$ to E_B is given by $|\{i \mid s \cdot e_i \geq r\}|$. Therefore, the curves $E^{5,\pm}$ have the smallest distance to E_B. Starting from the public key E_B, we thus first search the paths to the curves $E^{5,\pm}$. We do this by growing two neighborhoods (with positive and negative orientation) from E_B. Recall from Section 5.1 that the expected distance of the faulty curves is about $74/11 \approx 7$. But the distance from E_B to $E^{5,s}$ can be a lot larger (it is equal to $|\{e_i \mid s \cdot e_i = 5\}|$). Such large distances are rare: the probability of both $E^{5,\pm}$ having distance larger than 10 from E_B is, e.g., $\sum_{n=11}^{74-11} \sum_{m=11}^{74-n} \left(\binom{74}{n}\binom{74-n}{m} 9^{74-n-m}\right)/11^{74} \approx 0.3\%$. Hence, we do expect to find a connection to at least one of the curves $E^{5,\pm}$ within distance 10, meaning that we expect the first connection to cost no more than $2\sum_{i=0}^{10} \binom{74}{i} \approx 2^{40.6}$ isogeny step evaluations and likely less for at least some $H(E_t)$ in the neighborhood. From there, we will identify orientation for some primes, hence the search will be more efficient at each successive step because we need to search through fewer than 74 primes.

Example 10 (CTIDH-512). The faulty curves for *any* round in CTIDH are closer to the public key E_B than in the CSIDH case: it is 14 in the worst case (one prime per batch all having the same orientation) and the distance is 7 on average (Section 5.2). So the directed `pubcrawl` searches up to distance ≈ 7 (one with positive and one with negative orientation) are very likely to identify many of the hashed curves. Once we identify some faulty curves, we can identify other faulty curves quickly by small neighborhood searches thanks to the extra ordered structure of the CTIDH keyspace. We also benefit from the slightly increased probability of failure leading to more curves in the neighborhood of $E^{r,s}$.

Summary. In the hashed version, the main difference compared to the case in Section 5 is that we cannot mount a meet-in-the-middle attack starting from E_B and all faulty curves but can only search starting from E_B. Hence, we do not get the square-root speedup. Despite the increase in the costs it still possible to attack the hashed version.

Other sizes and variants work the same way with the concrete numbers adjusted. The brute-force searches to connect the effective round-r curves in large CSIDH versions do get very expensive but will still remain cheaper than the security level for average gaps between E_B and $E^{r,s}$ for the maximum r values.

8 Exploiting the twist to allow precomputation

In this section, we use quadratic twists and precomputation to significantly speed up obtaining the private key $\mathfrak{a}$ given enough samples E_t, especially for the “hashed” version described in Section 7.

Using the twist. The attack target is a public key $E_B = \mathfrak{a} * E_0$. Previously (Section 3), we attacked the computation of $\mathfrak{a} * E_0$ with disorientation faults. In this section, we will use E_{-B} as the input curve instead: Negating B is related to inverting $\mathfrak{a}$ because $E_{-B} = \mathfrak{a}^{-1} * E_0$. Moreover, applying $\mathfrak{a}$ to E_{-B} gives us back the curve E_0 and faulting this computation then produces faulty curves close to the fixed curve E_0. As E_{-B} is the *quadratic twist* of E_B, we will refer to this attack variant as *using the twist.*

The main trick is that twisting induces a symmetry around the curve E_0. This can be used to speed up `pubcrawl`: the opposite orientation of E_t (starting from E_0) reaches E_{-t}, so we can check two curves at once.

By precomputing a set $\mathcal{C}$ of curves of distance at most d to E_0, a faulty curve E_t at distance $d' \leq d$ is in $\mathcal{C}$ and can immediately be identified via a table lookup. Note that $\mathcal{C}$ can be precomputed once and for all, independent of the target instance, as for any secret key $\mathfrak{a}'$ the faulty curves end up close to E_0. The symmetry of E_{-t} and E_t also reduces storage by half.

Finally, this twisting attack cannot be prevented by simply recognizing that E_{-B} is the twist of E_B and refusing to apply the secret $\mathfrak{a}$ to such a curve: An attacker can just as easily pick a random masking value $\mathfrak{z}$ and feed $\mathfrak{z} * E_{-B}$ to the target device. The faulty curves E_t can then be moved to the neighborhood of

E_0 by computing $\mathfrak{z}^{-1} * E_t$ at some cost per E_t, or the attacker can precompute curves around $\mathfrak{z} * E_0$. The latter breaks the symmetry of E_t and E_{-t} and does not achieve the full speedup or storage reduction, but retains the main benefits.

Twisting CTIDH. The twisting attack is at its most powerful for CTIDH. As noted before, the sets $S^{r,\pm}$ are small in every round for CTIDH. The crucial observation is that in each round and for each orientation, we use at most one prime per batch (ignoring torsion noise, see Section 4.4). For a faulty curve E_t, the path $E_t \to E_0$ includes only steps with the same orientation and uses at most one prime per batch. With batches of size N_i, the total number of possible paths per orientation is $\prod_i (N_i + 1)$, which is about $2^{35.5}$ for CTIDH-512. Hence, it is possible to precompute *all* possible faulty curves that can appear from orientation flips from *any* possible secret key $\mathfrak{a}$.

Extrapolating the performance of `pubcrawl` (Section 6), this precomputation should take no more than a few core years. The resulting lookup table occupies $\approx 3.4\,\text{TB}$ when encoded naively, but can be compressed to less than $250\,\text{GB}$ using techniques similar to [38, § 4.3].

Twisting CSIDH. For this speed-up to be effective, the distance d we use to compute $\mathcal{C}$ must be at least as large as the smallest $|S^{r,\pm}|$. Otherwise, no faulty curves end up within $\mathcal{C}$. For CSIDH, the smallest such sets are $S^{r_{\max},\pm}$, where $r_{\max}$ is the maximal exponent permitted by the parameter; e.g., for CSIDH-512 $r_{\max} = 5$ and $S^{5,\pm}$ have an expected size ≈ 7. Precomputing $\mathcal{C}$ for $d \leq 7$ creates a set containing $\sum_{i=0}^{7} \binom{74}{i} \approx 2^{31}$ curves. Such a precomputation will either identify $S^{5,\pm}$ immediately, or allow us to find these sets quickly by considering a small neighbourhood of the curves $E^{5,\pm}$.

Note that for all the earlier rounds $r < r_{\max}$, the sets $S^{r,s}$ include $S^{r_{\max},s}$. Therefore, if we have the orientation s and have the set $S^{r_{\max},s}$, we can shift all the faulty curves by two steps for every degree in $S^{r_{\max},s}$. If we have misidentified the orientation, this shift moves the faulty curves in the wrong direction, even further away from E_0. This trick is particularly useful for larger r because eventually, many isogenies need to be applied in the shifts and we will have identified the orientation of enough primes so that the search space for `pubcrawl` becomes small enough to be faster.

Twisting in the hashed version. Precomputation extends to the hashed version from Section 7: we simply precompute $\mathcal{C}'$ which instead of E_t includes $H(E_t)$ for all E_t in the neighborhood of E_0.

Again, this works directly for attacking a hashed version of CTIDH and the effective round-$r_{\max}$ curves in CSIDH. To use precomputation for different rounds, one can replace the starting curve E_{-B} that is fed to the target device by the shift given exactly by the primes in $S^{r_{\max},s}$ (or, adaptively, by the part of the secret key we have already figured out). This has the same effect as above: shifting all the curves E_t with the same orientation *closer* towards E_0, hopefully so that the $H(E_t)$ are already in our database. If they are not then likely the opposite orientation appeared when we faulted the computation.

Summary. The benefit of using the twist with precomputation is largest for the hashed versions: we need a brute force search from E_0 in any case, and so we would use on average as many steps per round as the precomputation takes. For the non-hashed versions, the expensive precomputation competes with meet-in-the-middle attacks running in square root time. This means that in the hashed version we do not need to amortize the precomputation cost over many targets and have a clear tradeoff between memory and having to recompute the same neighborhood searches all over again and again.

9 Countermeasures

In this section, we present countermeasures against disorientation fault attacks from Section 3. We first review previous fault attacks on CSIDH and their countermeasures, as well as their influence on our attack in Section 9.1. We then discuss new countermeasures for one-point sampling from CSIDH and Elligator in Section 9.2, and estimate the costs of the countermeasures in Section 9.3.

9.1 Previous fault attacks and countermeasures

One way to recover secret keys is to target dummy isogenies with faults [10, 28]. Although these attacks are implementation-specific, the proposed countermeasures impact our attack too. Typically, real isogenies are computed prior to dummy isogenies, but the order of real and dummy isogenies can be randomized [10, 28] with essentially no computational overhead. When applied to dummy-based implementations, e.g., from [30, 32], this randomization means dummy isogenies can appear in different rounds for each run, which makes the definitions of the curves $E^{r,\pm}$ almost obsolete. However, we can instead simply collect many faulted round-1 samples. Each faulty curve E_t reveals a different set S_t due to the randomization, and with enough samples, a statistical analysis will quickly reveal all the $e_{i,j}$ just from the number of appearances among the sets S_t, again recovering the secret key.

Adapted to CTIDH, there are two possible variants of this randomization countermeasure: One could either keep the queue of real isogenies per batch as described above, but insert dummy isogenies randomly instead of at the end of the queue, or fully randomize the order of isogeny computations per batch including the dummy operations. In the first case, faulting round r if a dummy isogeny is computed in batch $\mathcal{B}_i$ means that no prime from this batch appears in the missing set. This effect is the same as missing torsion and thus our attack remains feasible. The net effect matches increased failure probabilities p_i and the larger neighborhoods simplify finding orientations. Note also that p_i is inflated more for batches with more dummy isogenies. In the second case when the entire queue is randomized, the same arguments as for CSIDH apply, and we can recover the secret key from statistical information with round-1 samples only.

Many fault attacks produce invalid intermediate values. In [10] some low-level protections for dummy isogenies to detect fault injections are proposed. This approach does not prevent our disorientation attack, and is orthogonal to our proposed countermeasures. Its performance overhead for the CSIDH-512 implementation from [32] is reported to be 7%.

Faulting memory locations can identify dummy isogenies [11]. In addition to the countermeasures above, the authors recommend using dummy-free implementations when concerned about fault attacks, with a roughly twofold slowdown [16]. However, as described in Section 5.3, dummy-free implementations are vulnerable to disorientation faults too.

Lastly, [10] reports that its fault attack theoretically could lead to disorientation of a point. Although the probability for this to happen is shown to be negligible, the authors propose to counter this attack vector by checking the field of definition of each isogeny kernel generator. This is rather expensive, with an overhead of roughly 30% for the implementation from [32], but also complicates the disorientation faults proposed in this work. We further discuss this in Section 9.2. We note that our countermeasures are significantly cheaper, but do not prevent the theoretical fault effect from [10].

9.2 Protecting square checks against fault attacks

The attack described in Section 3 can be applied to all implementations of CSIDH that use a call to `IsSquare` to determine the orientations of the involved point(s). The main weakness is that the output of `IsSquare` is always interpreted as $s = 1$ or $s = -1$, and there is no obvious way of reusing parts of the computation to verify that the output is indeed related to the x-coordinate of the respective point. For instance, faulting the computation of the Legendre-input $z = x^3 + Ax^2 + x$ results in a square check for a point unrelated to the actual x-coordinate in use, and yields a fault success probability of 50%.

Repeating square checks. One way to reduce the attacker's chances for a successful fault is to add redundant computations and repeat the execution of `IsSquare` k times. In principle, this means that the attacker has to fault all k executions successfully, hence reducing the overall fault success probability to $1/2^k$. However, if an attacker manages to reliably fault the computation of z or the Legendre symbol computation or to skip instructions related to the redundant computations, they might be able to circumvent this countermeasure.

Repeated square checks have been proposed for a different fault attack scenario [10]. There, `IsSquare` is used to verify the correct orientation for each point that generates an isogeny kernel. However, this countermeasure significantly impacts the performance of CSIDH, and could be bypassed as above.

Using y-coordinates. In CSIDH, the field of definition of the y-coordinate determines the orientation of a point. So, another simple countermeasure relying on redundant computation is to work with both x- and y-coordinates, instead of x-only arithmetic. We can then easily recognize the orientation of each point. But this leads again to a significant performance loss due to having to keep y-coordinates during all point multiplications and isogeny evaluations. We expect that this countermeasure is significantly more expensive than repeating `IsSquare` k times for reasonable choices of k.

Using pseudo y-coordinates. We propose a more efficient countermeasure: compute *pseudo* y-coordinates after sampling points. We sample a random x-coordinate and set $z = x^3 + Ax^2 + x$. If z is a square in $\mathbb{F}_p$, we can compute the corresponding y-coordinate $\tilde{y} \in \mathbb{F}_p$ through the exponentiation $\tilde{y} = \sqrt{z} = z^{(p+1)/4}$, and hence $\tilde{y}^2 = z$. Conversely, if z is a non-square in $\mathbb{F}_p$, the same exponentiation outputs $\tilde{y} \in \mathbb{F}_p$ such that $\tilde{y}^2 = -z$. Thus, as an alternative to `IsSquare`, we can determine the orientation of the sampled point by computing $z = x^3 + Ax^2 + x$, and the pseudo y-coordinate $\tilde{y}^2 = z^{(p+1)/4}$. If $\tilde{y}^2 = z$, the point has positive orientation, if $\tilde{y}^2 = -z$ it has negative orientation. If neither of these cases applies, i.e., $\tilde{y}^2 \neq \pm z$, a fault must have occurred during the exponentiation, and we reject the point.

This method may seem equivalent to computing the sign s using `IsSquare` as it does not verify that z has been computed correctly from x. But having an output value $\tilde{y} \in \mathbb{F}_p$ instead of the `IsSquare` output -1 or 1 allows for a much stronger verification step in order to mitigate fault attacks on the point orientation. We present the details of the original CSIDH algorithm including this countermeasure in Algorithm 2.

Steps 3 and 4 of Algorithm 2 contain our proposed method to determine the orientation s without using `IsSquare`. In order to verify the correctness of these computations, we add a verification step. First, we recompute z via $z' = x^3 + Ax^2 + x$, and in case of a correct execution, we have $z = z'$. Thus, we have $s \cdot z' = \tilde{y}^2$, which we can use as verification of the correctness of the computations of s, z, z', and $\tilde{y}$. If this were implemented through a simple check, an attacker might be able to skip this check through fault injection. Hence, we perform the equality check through the multiplications $X_Q = s \cdot z' \cdot X_{Q'}$ and $Z_Q = \tilde{y}^2 \cdot Z_{Q'}$, and initialize $Q = (X_Q : Z_Q)$ only afterwards, in order to prevent an attacker from skipping Step 8. If $s \cdot z' = \tilde{y}^2$ holds as expected, this is merely a change of the projective representation of Q', and thus leaves the point and its order unchanged. However, if $s \cdot z' \neq \tilde{y}^2$, this changes the x-coordinate X_Q/Z_Q of Q to a random value corresponding to a point of different order. If Q does not have the required order before entering the isogeny loop, the isogeny computation will produce random outputs in $\mathbb{F}_p$ that do not represent supersingular elliptic curves with overwhelming probability. We can either output this random $\mathbb{F}_p$-value, or detect it through a supersingularity check (see [4, 15]) at the end of the algorithm and abort. The attacker gains no information in both cases. The supersingularity check can be replaced by a cheaper procedure [10]: Sampling a

Algorithm 2: Evaluation of CSIDH group action with countermeasure

```
Input: $A \in \mathbb{F}_p$ and a list of integers $(e_1, \ldots, e_n)$.
Output: $B \in \mathbb{F}_p$ such that $\prod [\mathfrak{l}_i]^{e_i} * E_A = E_B$
 1: while some $e_i \neq 0$ do
 2:     Sample a random $x \in \mathbb{F}_p$, defining a point $P$.
 3:     Set $z \leftarrow x^3 + Ax^2 + x$, $\tilde{y} \leftarrow r^{(p+1)/4}$.
 4:     Set $s \leftarrow 1$ if $\tilde{y}^2 = z$, $s \leftarrow -1$ if $\tilde{y}^2 = -z$, $s \leftarrow 0$ otherwise.
 5:     Let $S = \{i \mid e_i \neq 0,\ \mathrm{sign}(e_i) = s\}$. Restart with new $x$ if $S$ is empty.
 6:     Let $k \leftarrow \prod_{i \in S} \ell_i$ and compute $Q' = (X_{Q'} : Z_{Q'}) \leftarrow [\frac{p+1}{k}]P$.
 7:     Compute $z' \leftarrow x^3 + Ax^2 + x$.
 8:     Set $X_Q \leftarrow s \cdot z' \cdot X_{Q'}$, $Z_Q \leftarrow \tilde{y}^2 \cdot Z_{Q'}$.
 9:     Set $Q = (X_Q : Z_Q)$.
10:     for each $i \in S$ do
11:         Set $k \leftarrow k/\ell_i$.
12:         Compute $R \leftarrow [k]Q$. If $R = \infty$, skip this $i$.
13:         Compute $\phi : E_A \to E_B$ with kernel $\langle R \rangle$.
14:         Set $A \leftarrow B$, $Q \leftarrow \phi(Q)$, and $e_i \leftarrow e_i - s$.
15: return $A$.
```

random point P and checking if $[p+1]P = \infty$ is much cheaper and has a very low probability of false positives, which is negligible in this case.

There are several ways in which an attacker may try to circumvent this countermeasure. A simple way to outmaneuver the verification is to perform the *same* fault in the computation of z and z', such that $z = z'$, but $z \neq x^3+Ax^2+x$. To mitigate this, we recommend computing z' using a different algorithm and a different sequence of operations, so that there are no simple faults that can be repeated in both computations of z and z' that result in $z = z'$. Faults in the computation of both z and z' then lead to random $\mathbb{F}_p$-values, where the probability of $z = z'$ is $1/p$.

The attacker may still fault the computation of s in Step 4 of Algorithm 2. However, this will now also flip the x-coordinate of Q to $-x$, which in general results in a point of random order, leading to invalid outputs. The only known exception is the curve $E_0\colon y^2 = x^3 + x$: In this case, flipping the x-coordinate corresponds to a distortion map taking Q to a point of the same order on the quadratic twist. Thus, for E_0, flipping the sign s additionally results in *actually* changing the orientation of Q, so these two errors effectively cancel each other in Algorithm 2 and the resulting curve is the correct output curve after all.

Protecting Elligator. Recall from Section 3 that two-point variants of CSIDH, including CTIDH, use the Elligator map for two points simultaneously, which requires an execution of `IsSquare` in order to correctly allocate the sampled points to P_+ and P_-.

We can adapt the pseudo y-coordinate technique from Section 9.2: we determine orientations and verify their correctness by applying this countermeasure for both P_+ and P_- separately. We dub this protected version of the Elligator sampling Elligreator. An additional benefit is that faulting the computations of the x-coordinates of the two points within Elligator (see [16, Algorithm 3]) is prevented by Elligreator.

In CTIDH, each round performs two Elligator samplings, and throws away one point respectively. Nevertheless, it is not known a priori which of the two points has the required orientation, so Elligreator needs to check *both* points anyway in order to find the point of correct orientation.

On the one hand, adding dummy computations, in this case sampling points but directly discarding some of them, might lead to different vulnerabilities such as safe-error attacks. On the other hand, sampling both points directly with Elligreator at the beginning of each round (at the cost of one additional isogeny evaluation) may lead to correlations between the sampled points, as argued in [3]. It is unclear which approach should be favored.

9.3 Implementation costs

Implementing this countermeasure is straightforward. While IsSquare requires an exponentiation by $(p-1)/2$, our pseudo y-coordinate approach replaces this exponent by $(p+1)/4$, which leads to roughly the same cost. (Note that neither has particularly low Hamming weight.) Furthermore, we require a handful of extra operations for computing z', X_Q, and Z_Q in Steps 7 and 8 of Algorithm 2. For the computation of z' we used a different algorithm than is used for the computation of z, incurring a small additional cost, for the reason discussed above.

Therefore, using this countermeasure in a 1-point variant of CSIDH will essentially not be noticeable in terms of performance, since the extra operations are negligible in comparison to the overall cost of the CSIDH action.

In 2-point variants, we use Elligreator, which requires two exponentiations instead of one as Elligator does. Thus, the countermeasure is expected to add a more significant, yet relatively small overhead in 2-point variants as in CTIDH. CTIDH uses two calls to Elligreator per round, and both executions contain two pseudo-y checks respectively.

We estimate the cost of our countermeasure in CTIDH-512. The software of [3] reports an exponentiation by $(p-1)/2$ to cost 602 multiplications (including squarings). Since CTIDH-512 requires roughly 20 rounds per run, we add two additional exponentiations by $(p+1)/4$ per round, and these have almost the same cost of 602 multiplications, the overhead is approximately $2 \cdot 20 \cdot 602 = 24080$ multiplications. Ignoring the negligible amount of further multiplications we introduce, this comes on top of a CTIDH-512 group action, which takes 438006 multiplications on average. Thus, we expect the total overhead of our countermeasure to be roughly 5.5% in CTIDH-512.

References

1. Gora Adj, Jesús-Javier Chi-Domínguez, and Francisco Rodríguez-Henríquez. Karatsuba-based square-root Vélu's formulas applied to two isogeny-based proto-

cols. Cryptology ePrint Archive, Paper 2020/1109, 2020. https://eprint.iacr.org/2020/1109.

2. Gora Adj, Jesús-Javier Chi-Domínguez, Víctor Mateu, and Francisco Rodríguez-Henríquez. Faulty isogenies: a new kind of leakage. Cryptology ePrint Archive, Paper 2022/153, 2022. https://eprint.iacr.org/2022/153.
3. Gustavo Banegas, Daniel J. Bernstein, Fabio Campos, Tung Chou, Tanja Lange, Michael Meyer, Benjamin Smith, and Jana Sotáková. CTIDH: faster constant-time CSIDH. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2021(4):351–387, 2021.
4. Gustavo Banegas, Valerie Gilchrist, and Benjamin Smith. Efficient supersingularity testing over $\mathbb{F}_p$ and csidh key validation. Cryptology ePrint Archive, Paper 2022/880, 2022.
5. Daniel J. Bernstein, Luca De Feo, Antonin Leroux, and Benjamin Smith. Faster computation of isogenies of large prime degree. In Steven D. Galbraith, editor, *Proceedings of the Fourteenth Algorithmic Number Theory Symposium*, pages 39–55. Mathematics Sciences Publishers, 2020. https://eprint.iacr.org/2020/341.
6. Daniel J. Bernstein, Mike Hamburg, Anna Krasnova, and Tanja Lange. Elligator: elliptic-curve points indistinguishable from uniform random strings. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *2013 ACM SIGSAC Conference on Computer and Communications Security, CCS'13, Berlin, Germany, November 4-8, 2013*, pages 967–980. ACM, 2013. https://eprint.iacr.org/2013/325.
7. Daniel J. Bernstein, Tanja Lange, Chloe Martindale, and Lorenz Panny. Quantum circuits for the CSIDH: optimizing quantum evaluation of isogenies. In Yuval Ishai and Vincent Rijmen, editors, *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*, volume 11477 of *Lecture Notes in Computer Science*, pages 409–441. Springer, 2019. https://eprint.iacr.org/2018/1059.
8. Ward Beullens, Thorsten Kleinjung, and Frederik Vercauteren. CSI-FiSh: Efficient isogeny based signatures through class group computations. In Steven D. Galbraith and Shiho Moriai, editors, *Advances in Cryptology - ASIACRYPT 2019 - 25th International Conference on the Theory and Application of Cryptology and Information Security, Kobe, Japan, December 8-12, 2019, Proceedings, Part I*, volume 11921 of *Lecture Notes in Computer Science*, pages 227–247. Springer, 2019.
9. Xavier Bonnetain and André Schrottenloher. Quantum security analysis of CSIDH. In Canteaut and Ishai [13], pages 493–522. https://eprint.iacr.org/2018/537.
10. 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 *17th Workshop on Fault Detection and Tolerance in Cryptography, FDTC 2020, Milan, Italy, September 13, 2020*, pages 57–65. IEEE, 2020. https://eprint.iacr.org/2020/1005.
11. Fabio Campos, Juliane Krämer, and Marcel Müller. Safe-error attacks on SIKE and CSIDH. In Lejla Batina, Stjepan Picek, and Mainack Mondal, editors, *Security, Privacy, and Applied Cryptography Engineering – 11th International Conference, SPACE 2021, Kolkata, India, December 10-13, 2021, Proceedings*, volume 13162 of *Lecture Notes in Computer Science*, pages 104–125. Springer, 2021.
12. Fabio Campos, Michael Meyer, Krijn Reijnders, and Marc Stöttinger. Patient zero and patient six: Zero-value and correlation attacks on csidh and sike. Cryptology ePrint Archive, Paper 2022/904, 2022.

13. Anne Canteaut and Yuval Ishai, editors. *Advances in Cryptology – EUROCRYPT 2020 – 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10-14, 2020, Proceedings, Part II*, volume 12106 of *Lecture Notes in Computer Science*. Springer, 2020.
14. Wouter Castryck and Thomas Decru. An efficient key recovery attack on SIDH (preliminary version). Cryptology ePrint Archive, Paper 2022/975, 2022.
15. Wouter Castryck, Tanja Lange, Chloe Martindale, Lorenz Panny, and Joost Renes. CSIDH: an efficient post-quantum commutative group action. In Thomas Peyrin and Steven D. Galbraith, editors, *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*, volume 11274 of *Lecture Notes in Computer Science*, pages 395–427. Springer, 2018. https://eprint.iacr.org/2018/383.
16. 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 Peter Schwabe and Nicolas Thériault, editors, *Progress in Cryptology – LATINCRYPT 2019 – 6th International Conference on Cryptology and Information Security in Latin America, Santiago de Chile, Chile, October 2-4, 2019, Proceedings*, volume 11774 of *Lecture Notes in Computer Science*, pages 173–193. Springer, 2019. https://eprint.iacr.org/2019/837.
17. Jorge Chávez-Saab, Jesús-Javier Chi-Domínguez, Samuel Jaques, and Francisco Rodríguez-Henríquez. The SQALE of CSIDH: square-root Vélu quantum-resistant isogeny action with low exponents. Cryptology ePrint Archive, Paper 2020/1520, 2020. https://eprint.iacr.org/2020/1520.
18. Jesús-Javier Chi-Domínguez and Francisco Rodríguez-Henríquez. Optimal strategies for CSIDH. Cryptology ePrint Archive, Paper 2020/417, 2020. https://eprint.iacr.org/2020/417.
19. Andrew M. Childs, David Jao, and Vladimir Soukharev. Constructing elliptic curve isogenies in quantum subexponential time. *J. Mathematical Cryptology*, 8(1):1–29, 2014. https://arxiv.org/abs/1012.4019.
20. John H. Conway and Neil J. A. Sloane. Low dimensional lattices VII: Coordination sequences. In *Proceedings of the Royal Society of London, Series A 453*, pages 2369–2389, 1997.
21. Jean-Marc Couveignes. Hard Homogeneous Spaces, 2006. IACR Cryptology ePrint Archive 2006/291. https://ia.cr/2006/291.
22. Christina Delfs and Steven D. Galbraith. Computing isogenies between supersingular elliptic curves over $\mathbb{F}_p$. *Des. Codes Cryptography*, 78(2):425–440, 2016. https://arxiv.org/abs/1310.7789.
23. Luca De Feo and Steven D. Galbraith. SeaSign: Compact isogeny signatures from class group actions. In Yuval Ishai and Vincent Rijmen, editors, *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 III*, volume 11478 of *Lecture Notes in Computer Science*, pages 759–789. Springer, 2019.
24. Luca De Feo and Michael Meyer. Threshold schemes from isogeny assumptions. In Aggelos Kiayias, Markulf Kohlweiss, Petros Wallden, and Vassilis Zikas, editors, *Public-Key Cryptography - PKC 2020 - 23rd IACR International Conference on Practice and Theory of Public-Key Cryptography, Edinburgh, UK, May 4-7, 2020, Proceedings, Part II*, volume 12111 of *Lecture Notes in Computer Science*, pages 187–212. Springer, 2020.

25. Alexandre Gélin and Benjamin Wesolowski. Loop-abort faults on supersingular isogeny cryptosystems. In Tanja Lange and Tsuyoshi Takagi, editors, *Post-Quantum Cryptography – 8th International Workshop, PQCrypto 2017, Utrecht, The Netherlands, June 26-28, 2017, Proceedings*, volume 10346 of *Lecture Notes in Computer Science*, pages 93–106. Springer, 2017.
26. Aaron Hutchinson, Jason T. LeGrow, Brian Koziel, and Reza Azarderakhsh. Further optimizations of CSIDH: A systematic approach to efficient strategies, permutations, and bound vectors. In Mauro Conti, Jianying Zhou, Emiliano Casalicchio, and Angelo Spognardi, editors, *Applied Cryptography and Network Security – 18th International Conference, ACNS 2020, Rome, Italy, October 19-22, 2020, Proceedings, Part I*, volume 12146 of *Lecture Notes in Computer Science*, pages 481–501. Springer, 2020. https://eprint.iacr.org/2019/1121.
27. Yi-Fu Lai, Steven D. Galbraith, and Cyprien Delpech de Saint Guilhem. Compact, efficient and UC-secure isogeny-based oblivious transfer. In Anne Canteaut and François-Xavier Standaert, editors, *Advances in Cryptology - EUROCRYPT 2021 - 40th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, October 17-21, 2021, Proceedings, Part I*, volume 12696 of *Lecture Notes in Computer Science*, pages 213–241. Springer, 2021.
28. Jason T. LeGrow and Aaron Hutchinson. (short paper) analysis of a strong fault attack on static/ephemeral CSIDH. In Toru Nakanishi and Ryo Nojima, editors, *Advances in Information and Computer Security - 16th International Workshop on Security, IWSEC 2021, Virtual Event, September 8-10, 2021, Proceedings*, volume 12835 of *Lecture Notes in Computer Science*, pages 216–226. Springer, 2021.
29. Luciano Maino and Chloe Martindale. An attack on SIDH with arbitrary starting curve. Cryptology ePrint Archive, Paper 2022/1026, 2022.
30. Michael Meyer, Fabio Campos, and Steffen Reith. On Lions and Elligators: An efficient constant-time implementation of CSIDH. In Jintai Ding and Rainer Steinwandt, editors, *Post-Quantum Cryptography – 10th International Conference, PQCrypto 2019, Chongqing, China, May 8-10, 2019 Revised Selected Papers*, volume 11505 of *Lecture Notes in Computer Science*, pages 307–325. Springer, 2019. https://eprint.iacr.org/2018/1198.
31. Michael Meyer and Steffen Reith. A faster way to the CSIDH. In Debrup Chakraborty and Tetsu Iwata, editors, *Progress in Cryptology – INDOCRYPT 2018 – 19th International Conference on Cryptology in India, New Delhi, India, December 9-12, 2018, Proceedings*, volume 11356 of *Lecture Notes in Computer Science*, pages 137–152. Springer, 2018. https://eprint.iacr.org/2018/782.
32. Hiroshi Onuki, Yusuke Aikawa, Tsutomu Yamazaki, and Tsuyoshi Takagi. (Short paper) A faster constant-time algorithm of CSIDH keeping two points. In Nuttapong Attrapadung and Takeshi Yagi, editors, *Advances in Information and Computer Security – 14th International Workshop on Security, IWSEC 2019, Tokyo, Japan, August 28-30, 2019, Proceedings*, volume 11689 of *Lecture Notes in Computer Science*, pages 23–33. Springer, 2019. https://eprint.iacr.org/2019/353.
33. Chris Peikert. He gives c-sieves on the CSIDH. In Canteaut and Ishai [13], pages 463–492. https://eprint.iacr.org/2019/725.
34. Damien Robert. Breaking SIDH in polynomial time. Cryptology ePrint Archive, Paper 2022/1038, 2022.
35. Alexander Rostovtsev and Anton Stolbunov. Public-key cryptosystem based on isogenies, 2006. IACR Cryptology ePrint Archive 2006/145. https://ia.cr/2006/145.

36. Élise Tasso, Luca De Feo, Nadia El Mrabet, and Simon Pontié. Resistance of isogeny-based cryptographic implementations to a fault attack. In Shivam Bhasin and Fabrizio De Santis, editors, *Constructive Side-Channel Analysis and Secure Design - 12th International Workshop, COSADE 2021, Lugano, Switzerland, October 25-27, 2021, Proceedings*, volume 12910 of *Lecture Notes in Computer Science*, pages 255–276. Springer, 2021.
37. Yan Bo Ti. Fault attack on supersingular isogeny cryptosystems. In Tanja Lange and Tsuyoshi Takagi, editors, *Post-Quantum Cryptography - 8th International Workshop, PQCrypto 2017, Utrecht, The Netherlands, June 26-28, 2017, Proceedings*, volume 10346 of *Lecture Notes in Computer Science*, pages 107–122. Springer, 2017.
38. Aleksei Udovenko and Giuseppe Vitto. Breaking the $ikep182 challenge, 2021. IACR Cryptology ePrint Archive 2021/1421.
39. Jacques Vélu. Isogénies entre courbes elliptiques. *Comptes Rendus de l'Académie des Sciences de Paris*, 273:238–241, 1971. https://gallica.bnf.fr/ark:/12148/cb34416987n/date.