# Graphs and cycles in curve-based collision walks

Sebastiano Boscardin, Jolijn Cottaar, and Monika Trimoska

Eindhoven University of Technology, The Netherlands
s.boscardin@tue.nl, jolijncottaar@gmail.com, m.trimoska@tue.nl

**Abstract.** In this work, we look at collision-based attacks both for elliptic curve and isogeny problems. Our main focus is on attacks in the multi-user setting, i.e., where one tries to recover all secret keys of a large set of $L$ public keys simultaneously, in a faster way than the total time required to attack each key individually. We consider three different problems: (1) the discrete logarithm problem on elliptic curves, (2) the isogeny pathfinding problem on supersingular $\mathbb{F}_{p^2}$-curves, (3) the isogeny pathfinding problem on supersingular $\mathbb{F}_p$-curves. We first survey the existing van Oorschot and Wiener style attacks on these three problems. Then we propose versions of the pseudorandom walks that are better suited for isogeny pathfinding problems without additional restrictions. We then apply these new walks to the multi-user setting and analyse which properties we can achieve compared to the discrete logarithm case where we show that we can extract all $L$ public keys with $L$ useful collisions and without an infiltrator (a user for which we know the secret key). Specifically, for (2) we can extract $L$ public keys with approximately $L + \log(L)$ useful collisions and no infiltrator needed. Our analysis shows however that heuristic and infiltrator-aided strategies are more efficient while enjoying practically similar gains. For (3) we can extract approximately $0.75L$ public keys with $L$ useful collisions, but with an infiltrator.

**Keywords:** multi-user, isogeny graphs, meet-in-the-middle, collisions, ECDLP.

## 1 Introduction

Birthday-based collision-search algorithms play a large role in cryptanalysis, both in symmetric and public key cryptography. Specifically for elliptic curve cryptography, it remains the state-of-the-art attack for the generic group case. In this work, we deal with parallel collision searches in the vOW framework, introduced by van Oorschot and Wiener [35]. This framework captures collision searches in pseudorandom walks in a parallelized setting. Trails of a defined pseudorandom mapping are computed until a collision is found, i.e., until two trails reach the same point. The collision can be detected because a portion of the traversed points, the so-called distinguished points, are stored in a shared memory. Versions of this collision search are used both in cryptanalysis of the Elliptic Curve Discrete Logarithm Problem (ECDLP) and in isogeny pathfinding

problems. We define and use *user-independent* walks, that is, walks where the random map is not defined in terms of the public key, but the public key figures only in the computation of the starting point. As an example, one can compare the original description of Pollard's rho random walk in Equation (2) and the user-independent version in Equation (4). Having a user-independent walk allows for using a precomputation step to aid in subsequently solving a concrete instance of the problem. Having the possibility of this precomputation approach is of course not interesting in the case where we need to recover the solution of only one instance to the problem [6]. This property however becomes useful in the multi-user setting.

When we talk about the *multi-user setting*, we mean the case where we need to find the secret keys of *all* users of a cryptosystem that share the same public parameters. There exists also the case where we are asked to solve *any* of the many problem instances. This case is less interesting to study as the goal would be to solve any arbitrary instance with some advantage over the fixed instance case. The *any* instance problem is however equivalent to the *one* instance problem for both the discrete logarithm case (see for instance, section 5.2 of [25]) and for the isogeny case (see for instance, Problem 3 [15] or Definition 2 [7]). On the other hand, in the case where we are asked to solve all given instances of the problem, the goal is to solve them in less time than a straightforward approach where we solve each instance one by one. With this approach, the time complexity is given by the product of the number of users and the estimated time complexity to solve one instance. The advantage is also expected to become non-negligible only when we have a large number of users.

Fouque, Joux, and Mavromati [20] in 2014 introduced a multi-user-style attack on the discrete logarithm problem using collision attacks in the vOW framework. In this multi-user setting we have access to a large number $L$ of public keys of users of a cryptosystem that share the same public parameters. Fouque, Joux, and Mavromati create a graph with vertices being the public keys of the users and create edges in between if we know some relation between the two public keys. We will work in this setting and call this relation graph the *multi-user* graph, to distinguish from other graphs we will consider. Then, after creating a certain number of edges (i.e., finding a certain number of collisions), one has a graph with a certain number of connected components. When one has an *infiltrator* (i.e., a public key with known secret key) in one such component, one can break all public keys in this component.

In this paper, we expand on their work by improving their result on the (elliptic curve) discrete logarithm problem and by considering how to realize a similar attack on isogeny-based problems. For the discrete logarithm case, we achieve the following features: (1) We can avoid producing useless collisions; (2) There is no need for an infiltrator in the algorithm; and (3) We recover the secret key of *all* users (deterministically). We then proceed by analyzing which of these features are present or can be obtained by tweaking the attacks in the isogeny setting as well. Specifically, we look both at the setting for isogenies defined over $\mathbb{F}_{p^2}$, when we have access to a large $\ell^k$-torsion with a sufficiently small

$\ell$, and isogenies defined over $\mathbb{F}_p$ as in CSIDH. Simply said, given two elliptic curves $E_0$ and $E_1$ find a path in the respective isogeny graphs between these two vertices. For the multi-user problem, $E_0$ is shared between all users (and in most applications it is the special curve with $j$-invariant 1728 that is often denoted as such) and this curve can be considered as an infiltrator.

The main ingredient needed for these multi-user attacks is finding collisions between public keys, which then leads to an edge in the multi-user graph. To create such collisions we need a vOW-type collision attack for isogeny graphs. The main attack that exists over $\mathbb{F}_{p^2}$ was originally targeted to SIDH [1] and operates under the assumption that there is an isogeny of given degree between the public key $E_1$ and a curve with known endomorphism ring $E_0$. It is a meet-in-the-middle style attack, which means that the collision is found at a vertex equally distant from $E_0$ and $E_1$. This means that this graph completely depends on the two input vertices and thus walking on this graph is not user-independent. This both hinders our goal of applying the approach to the multi-user setting effectively, but also hinders the option of using precomputations as in [5]. Furthermore, due to the need of a random function (usually just a hash function) in these types of attacks there is the possibility of hash collisions which lead to having to look for a so called 'golden collision'.

Similarly, in the $\mathbb{F}_p$ case, the only considered collision-type attacks are of a similar nature as in the $\mathbb{F}_{p^2}$ case and they have not been worked out for full size keys.

To remove the need for a fixed degree as well as the other considerations mentioned above (user-independency and the need for a golden collision), we will use, for both isogeny graphs, a pseudorandom walk which in its definition closely resembles the random walks used in the discrete logarithm case.

We then show how we can create the multi-user graph in a slightly different way than explained in [20] such that we need fewer collisions to be able to break more keys. This is made possible with the use of graph cycles.

### 1.1 Related work

We expand here on some related work in two directions.

**On the multi-user problem.** Two of the problems we aim to solve (namely Problem 4 and Problem 6) are special cases of the so called group action discrete logarithm for multiple instances which was analyzed by May and Ostuzzi in [26]. The authors follow a different approach than we will in this work based on precomputations to solve all $L$ instances with some constant probability in time $O(\sqrt{LN})$, with $N$ the order of the group acting (see [26, Corollary 1]). In this work, we obtain the same asymptotic computational complexity and we construct random walks in a similar fashion but we skip the precomputation phase. Instead, we build a collision graph and look at cycles and connected components. On the other hand, Problem 5 is different because, to the best of our knowledge, it cannot be formulated as a group action by a finite abelian group.

**On isogeny pathfinding.** Another method for isogeny pathfinding is the Delfs-Galbraith algorithm [17]. This algorithm until recently was considered to be the best known non-quantum algorithm for the general isogeny problem, since it has the same asymptotic complexity as collision-finding algorithms, but with negligible memory requirements. Furthermore, it does not need any additional assumptions, such as the knowledge of the degree. However, in spite of what its square-root complexity suggests at first glance, the Delfs-Galbraith algorithm does not fall into the birthday-based collision-search framework. For this reason, the algorithm can not benefit from using more memory (to the best of our attempts), and thus can not benefit from a precomputation or have a meaningful advantage in the multi-user setting - at least not as long as the number of users is smaller than $\sqrt{p}$. Hence, we do not treat it as part this work.

A groundbreaking attack on the general isogeny pathfinding problem was put forward during the write-up of this paper by Wesolowski [38]. The attack relies primarily on a theorem by Aubry, Oyono, and Vincent [3] stating that there exists an isogeny of degree at most $p^{\frac{1}{3}}$ between any curve $E$ and its Frobenius conjugate, denoted $E^{(p)}$. Using a smoothness heuristic on this degree and a meet-in-the-middle approach, the short isogeny can be found in time $O(p^{\frac{1}{3}+o(1)})$ and with equally large memory requirements. This isogeny, composed with the Frobenius isogeny that takes $E^{(p)}$ back to $E$, results in an endomorphism on $E$, which is enough to solve the problem [27]. This attack can be put into the vOW framework, but it is still not precomputation-friendly, or equivalently, multi-user-friendly, because of the following reason. Roughly, the algorithm gains an advantage by considering only curves that are at most a $p^{\frac{1}{6}}$-isogeny away from $E$ or $E^{(p)}$. Hence, any curve that we have stored for attacking one user is most likely far from the relevant search space of another user. It is however worthy to note that, up to a pretty high bound on the number of users, using Wesolowski's attack sequentially would outperform the results we obtain for the vOW multi-user algorithm for the $\mathbb{F}_{p^2}$ case. Another way in which the new attack downgrades our results is that the strategy we adopt for isogeny pathfinding for the $\mathbb{F}_{p^2}$ case without additional restrictions (Section 3.2) is outperformed, not only in memory complexity (as it already was by the Delfs-Galbraith algorithm), but also in time complexity.

Both of these algorithms fall outside of the vOW framework and their complexity analyses are not birthday-based, which is mainly why we consider them to be out of scope for this work. They are also not generic in the sense that they rely on some additional insights on the underlying mathematical structure. The Delfs-Galbraith algorithm relies on the occurrence density of curves defined over $\mathbb{F}_p$ in the supersingular isogeny graph, while Wesolowski's attack leverages the knowledge of having an unusually short isogeny between a curve and its Frobenius conjugate. This is not to be understood as the algorithms not being applicable in all variants of the problem. The relevant underlying structure is present and can be exploited in all cryptographic schemes based on isogenies, and the second algorithm is the current (and recently updated) state-of-the-art for defining security parameters.

We also look into doing van Oorschot-Wiener type attacks on CSIDH. Work by Chi-Domínguez, Esser, Kunzweiler, and May [12] have introduced an improved way of attacking small-key CSIDH, where the exponent is at most $\pm 2$ in the private key. The theoretical work they do is not applicable to full size keys, which is what we consider in this work.

### 1.2 Contributions

- In Section 3 we introduce new collision attacks for the isogeny pathfinding problems for both $\mathbb{F}_{p^2}$ and $\mathbb{F}_p$ cases. Specifically, these attacks will use user-independent walks and we do not need a large number of collisions to hopefully find the so-called "golden collision" but any valid collision gives us the secret we need.
- In Section 4 we analyse the multi-user attack in the setting of the discrete-logarithm problem, the $\mathbb{F}_{p^2}$ isogeny graph, and the $\mathbb{F}_p$ isogeny graph. The results can be found in Table 1. In particular, we did the following.
  - In the discrete logarithm case, we improve on the work of [20]. We reduce the number of collisions needed to exactly $L$, we remove the need for an infiltrator and we show that we can find all the keys instead of just a subset.
  - In the case of $\mathbb{F}_{p^2}$, we show that we can do a similar style of attack as in the discrete logarithm case. Our analysis shows that we can, in theory, achieve the same results with an additional $\log(L)$ collisions. This approach however contains a subexponential subroutine and many caveats. As such, we deem it less efficient than the simpler approach of using an infiltrator, and we are not confident in its practicality overall. We conclude that, contrary to the discrete logarithm case, we do need an infiltrator and we break about 75% of the secret keys.
  - Similarly, in the $\mathbb{F}_p$ case, we show that we do still need an infiltrator and we will be unable to break all the keys but a subset of them. With exactly $L$ collisions we expect to recover about 75% of the secret keys. Same as in [20], there is a natural trade-off between the number of extra collisions and the portion of secret keys we can find for both settings.

**Table 1.** The overview of what our multi-user attacks can achieve when attacking $L$ users.

| Problem | # needed collisions | Infiltrator | Number of keys recovered |
|---|---|---|---|
| ECDLP (Problem 4) | $L$ | $\times$ | $L$ |
| $\mathbb{F}_{p^2}$ (Problem 5) | $L$ | ✓ | $\approx 0.75L$ |
| $\mathbb{F}_p$ (Problem 6) | $L$ | ✓ | $\approx 0.75L$ |

## 2 Background

### 2.1 Preliminaries on elliptic curves and isogenies

We assume familiarity with elliptic curves and isogenies. We refer to [32] for background on elliptic curves and to [14] for a comprehensive overview of the use of isogenies in cryptography.

The points on an elliptic curve $E/\mathbb{F}_q$ along with the point at infinity form a group under addition. Let us denote the cardinality of this group by $N$. Thus, within this finite group we can define a discrete logarithm problem.

*Problem 1 (Elliptic Curve Discrete Logarithm Problem).*
Given two points $P, Q \in E(\mathbb{F}_q)$ for some prime power $q$ with $Q \in \langle P \rangle$, find the integer $a$ such that $Q = aP$.

We set $E_0 : y^2 = x^3 + x$, as usual. When looking at isogeny problems in this work, we only consider separable isogenies and supersingular elliptic curves. We will denote an isogeny as $\phi$ and its dual as $\hat{\phi}$. An endomorphism is an isogeny from an elliptic curve to itself. The set of all endomorphisms of an arbitrary elliptic curve $E$ form a ring and we will denote this as $\mathrm{End}(E)$. A known endomorphism ring $\mathrm{End}(E)$ can be pushed through an isogeny $\phi : E \to E'$ to find the endomorphism ring of $E'$.

When working in $\mathbb{F}_{p^2}$, we assume we have access to a large $\ell$-torsion. In practice, we will mostly work with a prime of the form $p = c2^e - 1$, for a large integer $e$ and a small cofactor $c$, as is done in many isogeny-based schemes like SQIsign [16] and PEGASIS [13], for instance[1]. Over $\mathbb{F}_{p^2}$ two curves are isomorphic if they have the same $j$-invariant. The isogeny graph we will consider thus is the $\ell$-isogeny graph where vertices are the isomorphism classes of elliptic curves and edges are $\ell$-isogenies. In this graph we can define a pathfinding problem.

*Problem 2 (Isogeny Pathfinding Problem over $\mathbb{F}_{p^2}$).*
Given two supersingular elliptic curves $E_0, E_1$ defined over $\mathbb{F}_{p^2}$ on an $\ell$-isogeny graph, find an efficient representation of an isogeny $\phi : E_0 \to E_1$.

Note that we choose here (for simplicity in the following) a definition of the problem where $E_0$ is a specific curve as previously defined. This is a special curve because, among other reasons, its endomorphism ring is known. The knowledge of the endomorphism ring is contagious for neighbouring curves, that is, curves to which we have an efficient isogeny representation from $E_0$. The general isogeny pathfinding problem is considered easy when the endomorphism ring of both input curves is known.

In the case of $\mathbb{F}_p$ we will consider the isogeny graph as used in CSIDH [9] and similar style schemes. In this context, it is also relevant to introduce the algebraic view of our setting. The ring of endomorphism over $\mathbb{F}_p$ is commutative

[1] This is only required when the *kernel isogeny representation* is used and not, for instance, when we are using modular polynomials for the isogeny walks in the final variant. We still make this assumption in general to keep the exposition uniform.

and isomorphic to an order $\mathcal{O}$ in an imaginary quadratic field $\mathbb{Q}(\pi)$, for $\pi$ the Frobenius endomorphism. For this exposition, we will use the CSIDH [9] variant where we take $p \equiv 3 \mod 4$ and the isomorphism classes of supersingular elliptic curves over $\mathbb{F}_p$ with endomorphism ring $\mathcal{O} = \mathbb{Z}[\sqrt{-p}]$. The set of invertible ideals modulo non-zero principal ideals in $\mathcal{O}$ defines an ideal-class group, denoted $cl(\mathcal{O})$, and we denote the class of the ideal $\mathfrak{a}$ by $[\mathfrak{a}]$. The class group acts in a free and transitive manner on the set of supersingular elliptic curves defined over $\mathbb{F}_p$ modulo isomorphism whose ring of $\mathbb{F}_p$-endomorphisms is isomorphic to $\mathcal{O}$. We will denote this set by $\mathcal{E}$, and the action of $[\mathfrak{a}]$ on $E \in \mathcal{E}$ by $[\mathfrak{a}]E$. We represent each elliptic curve defined over $\mathbb{F}_p$ modulo isomorphism, by its Montgomery coefficient $A \in \mathbb{F}_p$.

The isogenies that we are able to compute using Vélu's formulas depend on the smoothness of $p+1$. For a fixed prime $p$ of the form $p = 4 \cdot \ell_1 \cdots \ell_n - 1$ where the $\ell_i$ are distinct odd primes, we are able to compute $\mathbb{F}_p$-rational isogenies of degrees $\ell_i$. Another convenient feature of this setting is that all of the primes $\ell_i$ split in $\mathbb{Z}[\sqrt{-p}]$ as $(\ell_i) = (\ell_i, \sqrt{-p} - 1)(\ell_i, \sqrt{-p} + 1)$. We denote these factors by $\mathfrak{l}_i$ and $\overline{\mathfrak{l}_i}$, as they are complex conjugates of each other, and we denote their respective equivalence classes by $[\mathfrak{l}_i]$ and $[\mathfrak{l}_i]^{-1}$. Hence, all ideal classes can be written as

$$[\mathfrak{a}] = [\mathfrak{l}_1]^{e_1}[\mathfrak{l}_2]^{e_2} \cdots [\mathfrak{l}_n]^{e_n},$$

where the $e_i$ can be either positive or negative exponents. Note that in this setting, the order and structure of the class group are unknown, so it is uncertain whether the ideals $\prod_{i=1}^{n} \mathfrak{l}_i^{e_i}$ cover the entire class group. This results in what is known as a Restricted Effective Group Action (REGA) in the literature [2]. Then, the element $[\mathfrak{l}_i]$ acting on $\mathcal{E}$ corresponds to computing a *positive* $\ell_i$-isogeny, while $[\mathfrak{l}_i]^{-1}$ acting on $\mathcal{E}$ corresponds to a *negative* $\ell_i$-isogeny. Computationally, to do a positive $\ell_i$ isogeny from a curve $E$, we need an $\mathbb{F}_p$-rational point of order $\ell_i$ as input to Vélu's formulas. Conversely, to compute a negative $\ell_i$-isogeny, we need to choose a point that is not $\mathbb{F}_p$-rational, or rather, a point that is $\mathbb{F}_p$-rational and of order $\ell_i$ on the twist of $E$. Furthermore, the set of exponents is chosen within a bound $[-m; m]$, where $m$ is the smallest integer satisfying $(2m+1)^n \geq \sqrt{p}$, to ensure that the keyspace is big enough for the required security level. However, for cryptanalysis of any scheme whose security relies on this problem, it is enough to find any *equivalent* key. An equivalent key here means any isogeny connecting the input curves, even if it is not a CSIDH key as described in this paragraph, i.e. even if the exponents of the primes are not within the predefined bound.

*Problem 3 (Isogeny Pathfinding Problem over $\mathbb{F}_p$).*
Given two supersingular elliptic curves $E_0$ and $E_1$ defined over $\mathbb{F}_p$, find a vector $(e_1, \ldots, e_n) \in \mathbb{Z}^n$ such that $E_1 = [\mathfrak{a}]E_0$ for some $[\mathfrak{a}] = [\mathfrak{l}_1]^{e_1} \cdots [\mathfrak{l}_n]^{e_n}$.

### 2.2 Preliminaries on graphs

We will first look at some general results on directed multigraphs. We will denote these graphs as $G(V, A)$, with $V$ the set of vertices and $A$ the multiset of arcs,

i.e., ordered pairs of elements of $V$. We use the concept of weakly connected components, as in, the components are connected in an undirected fashion.

**Lemma 1.** *[4, Remark 1.1.2] In the graph $G = (V, A)$, the number $B$ of independent cycles and the number $C$ of (weakly) connected components are related by the equation*

$$B + |V| = C + |A| \tag{1}$$

A simple corollary following from this is that if a graph has at least one more arc than the number of vertices, then there is a (weakly) connected component with 2 cycles.

**Corollary 1.** *If a graph $G = (V, A)$ has $L$ vertices and at least $L+1$ arcs, there is a (weakly) connected component of $G$ with two independent cycles.*

*Proof.* From the previous lemma we have

$$\begin{aligned} B + L &= C + |A| \\ B + L &\geq C + L + 1 \\ B &\geq C + 1. \end{aligned}$$

So we see that the number of independent cycles $B$ is at least one more than the number of connected components $C$. Hence, there is at least one connected component with two independent cycles. □

Now let us introduce the specific graph we will mostly be working with, the functional graph.

**Definition 1.** *A directed multigraph $G(V, A)$ is a functional or successor graph if each vertex has outdegree 1, i.e., for each vertex $v \in V$ there exists exactly one element $(v, -) \in A$.*

The expected size of the largest connected component and the expected number of connected components of a functional graph are well known.

**Lemma 2.** *Consider the set of functional graphs (Definition 1), i.e., directed multigraphs with each edge having out-degree exactly 1, equipped with the uniform probability distribution. We know the following:*

- *The expected size of the largest connected component of a functional graph is asymptotically close to $0.76|V|$ [19, Theorem 8].*
- *The expected number of connected components of a functional graph is $\frac{1}{2}\log(|V|) + o(1)$ [24].*

**Lemma 3.** *Let $G = (V, A)$ be a functional graph. Then all (weakly) connected components of $G$ contain a non-trivial cycle.*

*Proof.* We argue by contradiction. Suppose there exists a (weakly) connected component in $G$ that does not contain a non-trivial cycle, say $G_1 = (V_1, A_1)$. This component needs to contain at least one vertex, say $v_1 \in V_1$. Because $G$ is functional there must exist an arc from $v_1$ to some vertex $v_2 \in V_1$, that is $(v_1, v_2) \in A_1$. This $v_2$ cannot be $v_1$ itself, since this would create a non-trivial cycle, hence contradiction. Thus, there are $|V_1| - 1$ choices for $v_2$. Again, there needs to be an arc between $v_2$ and some other vertex, which cannot be $v_2$ itself or $v_1$ since that would create a non-trivial cycle. Thus, there are $|V_1| - 2$ options left. Since $V_1$ is a finite set, at some point we reach $v_{|V_1|}$, which has no options left except to connect to some vertex already visited, hence creating a cycle. Thus, we can conclude by contradiction that our connected component must have a cycle. It is not-trivial because, by construction, it never repeats an arc. □

An alternative proof can be derived from Lemma 1. In fact, a functional graph has as many vertices as arcs in every connected component. Then we apply Equation (1) to a connected component and since $C = 1$ we also find $B = 1$.

### 2.3 The multi-user setting

We now define the following three problems representing the multi-user equivalent of the problems introduced in Section 2.1. Here and in the following, the number of users is denoted by $L$.

*Problem 4 (Multi-user Elliptic Curves Discrete Logarithm Problem).*
Given $L+1$ points $P, Q_1, \ldots, Q_L \in E(\mathbb{F}_q)$ for some prime power $q$ with $Q_i \in \langle P \rangle$ for $i \in \{1, \ldots, L\}$, find integers $a_1, \ldots, a_L$ such that $Q_i = a_i P$.

*Problem 5 (Multi-user Isogeny Pathfinding Problem over $\mathbb{F}_{p^2}$).*
Given $L+1$ supersingular elliptic curves $E_0, E_1, \ldots, E_L$ defined over $\mathbb{F}_{p^2}$ find $L$ efficient representations of isogenies $\phi_1, \ldots, \phi_L$ such that $\phi_i : E_0 \to E_i$.

*Problem 6 (Multi-user Isogeny Pathfinding Problem over $\mathbb{F}_p$).*
Given $L+1$ supersingular elliptic curves $E_0, E_1, \ldots, E_L$ defined over $\mathbb{F}_p$, find vectors $(e_{1,1} \ldots, e_{1,n}), \ldots, (e_{L,1} \ldots, e_{L,n}) \in \mathbb{Z}^n$ such that $E_i = [\mathfrak{a}_i]E_0$ with $[\mathfrak{a}_i] = [\mathfrak{l}_1]^{e_{i,1}} \cdots [\mathfrak{l}_n]^{e_{i,n}}$.

The multi-user setting for discrete logarithms was first treated by Kuhn and Struik in [25], where they explain how to use the first $k$ discrete logarithm solutions and the collected distinguished points to speed up the computation of the $(k+1)$th discrete log[2]. As per the van Oorschot and Wiener technique [36], random walks are used to find distinguished points. The points fall in one of $(L+1)$ disjoint sets, namely the *public* set, containing distinguished points from walks that start at some $Q_a = aP$ for which we know $a$, and the $L$ *secret* sets (one for each user), containing points from walks that start at $Q_i + Q_b$, where $Q_i$ is the public key of the $i$th user and $Q_b = bP$ for some known $b$. When there is

[2] A prior mention of the idea is attributed to Silverman and Stapleton.

a collision between a point in the public set with a point from one of the secret sets, the discrete logarithm of one user is recovered and the corresponding secret set becomes public. The analysis assumes that the discrete logs for different users are found sequentially. The speedup comes from the fact that the public set of points can be reused and gets increasingly larger for each succeeding user. In subsequent work, Fouque, Joux, and Mavromati [20] introduce the graph connectivity approach which allows for better parallelization and a simpler analysis. They define the (called here) multi-user graph where vertices denote the public keys of users and the edges denote collisions found between the users. Again, a collision being between user $i$ and $j$ entails that one of the distinguished points in the collision is from a walk starting from a linear combination of $P$ and $Q_i$, while the other is from a linear combination of $P$ and $Q_j$. Hence, an edge in the graph represents a known linear relation between $a_i$ and $a_j$ (notations as in Problem 4). Similarly to computing a public set of points in [25], a few trails are computed starting from random points for which the discrete logarithm (base $P$) is known. These trails correspond to new vertices in the graph, and we will refer to them as *infiltrators*[3]. The analysis relying on random graph properties shows that when there are approximately $cL/2$ edges created, for a small constant $c$, the graph has a large connected component containing an infiltrator with high probability. As an example, for $c = 4$ it is expected that $0.98L$ vertices are in the large component. Then, the discrete logarithm of all users whose vertices fall in the large connected component is recovered. Both approaches result in an asymptotic time complexity of $\tilde{\mathcal{O}}(\sqrt{LN})$, while the Kuhn-Struik method has the additional requirement that $L \ll \sqrt[4]{N}$.

## 3 Variants of the vOW collision search

In this section we start by explaining the general collision search approach by van Oorschot and Wiener [36]. We then explain how we find a collision in the ECDLP case, which falls under the *small number of collisions* case in the original work. We then start looking at collision-finding for the isogeny graphs defined over $\mathbb{F}_{p^2}$ or $\mathbb{F}_p$. For both, we first explain the current state-of-the-art algorithms that are intended for isogeny pathfinding with some additional information, and fall under the meet-in-the-middle variant of the collision search that is intended for finding a *large number of collisions* in the original work. We then propose alternative versions of the van Oorschot and Wiener (vOW) collision attack that mimic the discrete-log variant, and we explain why this is a better choice when looking at the corresponding path-finding problems without additional restrictions.

The general vOW framework is intended for cryptographic problems that can be reduced to the problem of finding a *collision*.

[3] There is a small number of them and they are neglected in the analysis. Hence, the graph is still considered to have exactly $L$ vertices.

*Problem 7.* Let $S$ be a finite set of cardinality $N$, let $f : S \to S$ be an efficiently-computable semi-random function. Find a collision for $f$, or in other words find $x$ and $x'$ such that $f(x) = f(x')$, where $x \neq x'$.

In their 1999 paper [36], van Oorschot and Wiener introduce a collision-finding algorithm which is well suited for parallelization. The main motivation is applying the Pollard's rho-method [31] in a multi-core setting and making use of the available memory to make the collision discovery more efficient than in a sequential method. In the original work, the ideas were applied to computing discrete logarithms, finding hash function collisions, and performing meet-in-the-middle attacks on double encryption.

The method to solve Problem 7 for both the discrete logarithm variant and the meet-in-the-middle variant, goes as follows. We use an efficiently-computable semi-random $f$ to create trails in the set $S$ until we reach a so-called *distinguished point*[4]. These points are elements of $S$ with an easily testable distinguishing property, for example, having $k$ trailing zero bits in some representation of the point. The fraction of distinguished points in $S$ will be denoted by $\theta$, whose value depends on the chosen distinguishing property. In the example above, we have $\theta = \frac{1}{2^k}$. In the parallel setting, each thread starts at a random element of $S$ and keeps applying $f$ until a distinguished point is reached (or a certain number of steps are done without finding a distinguished point since one can get stuck in a cycle). Then, if a distinguished point is reached, the starting point and the distinguished point are stored in memory, where indexing is based on the distinguished point. A core property for obtaining a theoretically perfect parallelization speedup is that $f$ is a random but deterministic function, meaning that the same input always leads to the same output. In return, collisions that occur between non-distinguished (and hence, not stored) points are discovered the first time a distinguished point is reached. Thus, the birthday paradox applies and the expected number of steps before a collision is found is given by the following lemma, which can be directly applied to derive the complexity of the algorithm, taken as the number of $f$-evaluations.

**Lemma 4.** *[36, Lemma 1] The expected number of steps of the random walk defined by the iteration of $f$ before a collision occurs (as defined in Problem 7) is asymptotic to*

$$\sqrt{\frac{\pi N}{2}}.$$

The main difference in this approach between the two variants is the choice of a graph that the trails are created in. In the discrete logarithm variant we walk in essence in the same graph as where the elements we try to attack live in. While for the meet-in-the-middle variant, as the name suggests, we work only with the points that lie in the middle of a meet-in-the-middle-style attack. These usually consist of elements that are a certain known distance in the graph away

[4] We continue using the term *point* on the random walk, even if it is not necessarily a point on an elliptic curve.

from the elements we are trying to find a map between. These types of attacks are usually used when the existence of a path of certain length is guaranteed. The downside is that many collisions exist for the function $f$, but we are looking for the *golden* collision, that is, the collision which leads to a solution to the cryptanalytic problem at hand. This is a result of how the $f$ function is defined, which will be detailed later. Note finally that just finding two trails that end at the same distinguished point is not enough to find the actual collision (like in the discrete logarithm variant), but one would have to recompute part of the trails to find where the collision first occurred.

### 3.1 Elliptic Curve Discrete Logarithm

In this section we will explain the discrete logarithm variant of vOW, applied to solving Problem 1. The collision approach for solving the discrete logarithm problem aims to find two different linear combinations of $P$ and $Q$ that result in the same point. Having $a_1P + b_1Q = a_2P + b_2Q$, we can compute $x = \frac{a_1-a_2}{b_2-b_1} \mod (\text{ord}(P))$ if $(b_2 - b_1, \text{ord}(P)) = 1$. To find such a collision, the random walk starts from a point $a_0P + b_0Q$ and the $f$ mapping is defined such that on input a linear combination of $P$ and $Q$ outputs a point that is also a linear combination of $P$ and $Q$, updating the $a, b$ coefficients. Let $G$ be the group generated by $P$. A folklore choice for defining the $f$ function is, for any point $R \in G$

$$f(R) = \begin{cases} R + P & \text{if } R \in S_1 \\ 2R & \text{if } R \in S_2 \\ R + Q & \text{if } R \in S_3. \end{cases} \tag{2}$$

where $\{S_1, S_2, S_3\}$ is a chosen partition of the group $G$. When a trail arrives to a distinguished point, the coefficients of the starting point are stored in a data structure indexed by the $x$-coordinate of the distinguished point for faster collision detection. The algorithm stops when the first collision is found. The only exception for a collision being useful is if we have $(b_2 - b_1, \text{ord}(P)) \neq 1$, which happens with negligible probability if $\text{ord}(P)$ is prime. The two trails of the collision are then recomputed from the starting point to derive the $a, b$ coefficients of the distinguished points and compute $x$. The complexity of the approach follows directly from Lemma 4, adding $\frac{1}{\theta}$ for the number of steps to discover the collision once it occurs. The memory complexity of the approach is $\theta\sqrt{\frac{\pi N}{2}}$ because it only stores a fraction $\theta$ of all the visited points.

### 3.2 In the $\mathbb{F}_{p^2}$ isogeny graph

Now let us see how we can solve Problem 2, i.e., find an isogeny between two arbitrary curves $E_1$ and $E_2$. We start by introducing the meet-in-the-middle variant of the vOW attacks introduced by Adj, Cervantes-Vázquez, Chi-Domínguez, Menezes and Rodrígeuz-Henríquez [1], which focuses on attacking the underlying problem of the SIDH scheme [22] but does not use the knowledge of torsion point

information. Note that in this case the goal is to find a fixed-degree isogeny, say $\ell^e$, for some integer $e$. In the case of SIDH, $\ell$ would either be 2 or 3, and since 2-isogenies are computationally cheaper the attack focused on that case.

**The meet-in-the-middle variant.** The elements over which the trails walk are elliptic curves that lie exactly a $2^{e/2}$-isogeny away from either $E_0$ or $E_1$. A collision between two trails in this case does not necessarily mean that we have found an isogeny between $E_0$ and $E_1$, since the collision might be between two elliptic curves that are both connected to either $E_0$ or $E_1$. This is a first of two reasons why not any collision can serve as the golden collision. Starting points of the trails are created by picking either $E_0$ or $E_1$ randomly and taking a random $2^{e/2}$-isogeny. Then $f$ will be applied on this point until a distinguished point is reached.

To define $f$ we need the following notation.

- $\mathcal{A}_s$ for $s \in 0, 1$ denotes the set of all order-$\ell^{e/2}$ cyclic subgroups of $E_s[\ell^{e/2}]$.
- $I = \{1, ..., \ell^{e/2-1}(\ell + 1)\}$ is a set with the same cardinality as $\mathcal{A}_0$ and $\mathcal{A}_1$. Hence, it is a set of indices for $\mathcal{A}_0$ and $\mathcal{A}_1$.
- $S = 0, 1 \times I$ is how we define the domain and codomain of $f$.

We define a function $f : S \to S$, by using the following functions.

1. $g_s : I \to \mathcal{A}_s$, which bijectively maps the $i \in I$ to its corresponding element of $\mathcal{A}_s$.
2. $\psi_s : \mathcal{A}_s \to \mathbb{F}_{p^2}$, which computes the isogeny induced by taking the element of $\mathcal{A}_s$ (i.e., the corresponding subgroup of order $\ell^{e/2}$) from $E_s$ and outputs the $j$-invariant of its codomain.
3. $H : \mathbb{F}_{p^2} \to \{0, 1\}^*$, which hashes the found $j$-invariant and outputs a pseudorandom bit-string.
4. $w : \{0, 1\}^* \to S$ any encoding function which takes a bit-string and outputs a side $s$ and an index $I$.

Then $f$ is defined as:

$$f(s, i) = w \circ H \circ \psi_s \circ g_s(i).$$

A point in $S$ is distinguished when a fixed number of the least significant bits of the hash of this point is below some threshold. When two trails end at the same distinguished point, we first check if one of the trails did not start in the other trail (called a Robin-Hood collision). We then find the actual collision by walking to the points on the trail until the trails merge. An important note to make here is that, contrary to the discrete logarithm variant, a collision does not always lead to a break. For example, a collision can be found between two points with the same $s$, in other words we find two maps from $E_s$ to the same point in the middle[5]. Or another option is that a collision can be found due

[5] This might give us a useful endomorphism, but this case was not used in the original proposal.

to the use of the function $H$. In practice this is usually just a hash function. A collision in the hash function $H$ yields a collision in the composition $f$ and occurs because there are two different $j$-invariants that hash to the same bit-string. However, recomputing the trails to recover the collision, we will find that the the two trails do not point to the same elliptic curve. So it is important to keep searching until a certain golden collision is found, that is, one that gives two elements $(0, i_0) \in S$ and $(1, i_1) \in S$. We can find an isogeny from $E_0$ to $E_1$ by applying $\psi \circ g$ to the found elements and combining the output into an isogeny. For specifics on runtimes and more specific on how to implement see [1].

**A discrete logarithm variant.** Now we look at Problem 2 as described, so we assume arbitrary degree. We would like to finalize the attack with *any* collision and not wait for the elusive golden collision. To this end, we analyze why in the previous algorithm not all collisions are golden collisions. The first reason, recall, is that a collision between two trails starting from the same curve is not useful. This collision however, yields an endomorphism and if it concerns a curve with unknown endomorphism ring, one can use it to recover it and solve the problem in this manner [27]. More importantly, a collision can be due to a collision in the hash function. In fact, we see that $w \circ H$ maps from a $j$-invariant to $S$, so from a domain of size $\approx p^2$ to a set of size $2(\ell^{e/2-1}(\ell+1))$. This discrepancy produces a lot of hash collisions that are not real collisions. Recall that the obtained trade-off is that we are significantly reducing the search space to only the curves *in the middle*.

For the unrestricted variant of the problem, we have as search space the entire isogeny graph. Actually, the assumption on $p$ having a large power of $\ell$ factor allows us to restrict to the entire $\ell$-isogeny graph. Since we are walking through the entire search space, we can hope to build the walk in the discrete logarithm setting of the vOW attack. To truly mimic the discrete logarithm setting, we need a random walk that not only considers as input *where we are* to *decide* where we need to go, but also to *compute* where we go. This is best seen in Pollard's walk in Equation (2) by noticing that the point $R$ figures both in the case distinction and in the output computation.

Our second goal is to introduce a function $f$ that is not dependent on the user, i.e., none of the user-dependent characteristics are an input to the function. This is with the goal to have a precomputation-friendly attack, or more importantly for our purposes, to have an attack that adapts well to the multi-user setting.

We create trails in the $\ell$-isogeny graph over $\mathbb{F}_{p^2}$, which we denote by their $j$-invariants. Thus, a collision gives us an $\ell^r$-isogeny, but now $r$ is an arbitrary integer. It is even possible that $r > e$. Starting points are created by randomly choosing between $E_0$ and $E_1$ and then taking a random $\ell^t$ isogeny, where $t$ is dependent on the size of $p$ and chosen such that this isogeny is sufficiently randomized. In the case of $p = c2^e - 1$ and $c$ odd, we let $\ell = 2$, since it is cheap and we have acces to a large torsion. Since the 2-isogeny graph is Ramanujan we choose $t = 2\log_2(p) \approx e$. This ensures that a random isogeny of degree $2^t$ from any given curve ends in a supersingular curve with distribution close to uniform.

To create a pseudo-random function $f$ in this $\ell$-isogeny graph, without backtracking, we use modular polynomials. We find the modular polynomial for our current location by evaluating $\Phi_\ell(j_i, X)$ and we factor out the $j$-invariant of the last visited node to make sure there is no backtracking. Then the resulting polynomial $\frac{\Phi_\ell(j_i,X)}{X-j_{i-1}}$ is factored and the remaining $\ell$ roots are deterministically ordered. The $(j_i \mod \ell)$th root is chosen which points to the chosen $\ell$-isogeny. Thus $f(j_i)$ simply follows these steps.

This function is familiar as the CGL hash function [11]. So after all the analysis on the properties we require from the function for use in the multi-user setting, we fall back to a folklore strategy whose elements appear as early as the works of Kohel [23] and Galbraith [21]. In [23], such walks are used to build a spanning tree until the same node is obtained twice, with stated worst case complexity of $\mathcal{O}(p)$, and in [21], it is used to solve pathfinding between ordinary curves with the same number of points defined over $\mathbb{F}_p$. A mention for the supersingular case states a complexity of $\mathcal{O}(p \log(p))$. Finally, the CGL proposal looks at the algorithm as computing random walks on optimal expander graphs with the up-to-date complexity estimate of $\mathcal{O}(\sqrt{p})$. We are able to plug these ideas into the vOW framework in a straightforward manner.

This function does not necessarily behave as a random function, depending on the size of $\ell$, which directly yields the number of cases. For instance, Teske [33] showed experimentally that the walk proposed by Pollard originally performs on average slightly worse than a random walk, which prompted her to propose the additive and mixed walks. Experiments in [33] show that if we define $f$ as a function with about 20 cases, additive walks are close to random walks. Similarly here, when experimenting with $\ell = 2$ we do not expect to reach the birthday bound. We can however do our analysis with steps of length 5 and see that as one function evaluation. Then we would have $2^5$ different options, since we do not allow backtracking. Our initial experiments show that with this number of cases we approach the birthday bound. This modification is however only for analysis purposes. In practice, the fastest way to do these 5 steps is by computing them one by one. Since checking the distinguished property is computationally cheap, it is much more efficient to consider the intermediate steps as well, even if we then do not completely adhere to the birthday bound. We report these experiments in Table 2. Finally, points are easily distinguishable by some property of the

**Table 2.** Analysis on number of $f$ evaluations needed to find one collision in the $F_{p^2}$ graph, average of 500 runs.

| Prime | # bits | Number of steps | Number of $f$ evaluations |
|---|---|---|---|
| 5119 | 13 | 1 | 32 |
| | | 5 | 15.6 |
| 917503 | 20 | 1 | 428.5 |
| | | 5 | 92.7 |

$j$-invariant, for example, an appropriate number of trailing zeros. We point out that these choices could not have been made for the $f$ function for the fixed-degree case, since then we would not be able to benefit from the promise of a degree which is significantly smaller than the expected (on average) smallest-degree isogeny between two arbitrary curves.

### 3.3 In the $\mathbb{F}_p$ isogeny graph

Similar to the $\mathbb{F}_{p^2}$ case, we will start by looking at the meet-in-the-middle variant which is most commonly cited when discussing CSIDH security [12]. We will consider a few variants of this version, since this has not been studied when working with full size keys. There is some theoretical work in small key CSIDH [12], which only works with keys of exponent size up to $\pm 2$. In this work we assume one uses full size keys up to $\pm 5$ and the work of [12] does not extend in a straightforward way to these sizes. We will then introduce our discrete logarithm variant of vOW in the $\mathbb{F}_p$ isogeny graph and compare the different attacks we have.

**The meet-in-the-middle variant.** The meet-in-the-middle variant works similar to the version in the $\mathbb{F}_{p^2}$ case. A collision in the $\mathbb{F}_p$ graph possibly gives us an ideal $[\mathfrak{a}]$ such that $[\mathfrak{a}]E_0 = E_1$ for a given $E_0$ and $E_1$. In isogeny based cryptography, $E_0$ and $E_1$ are public while the connecting ideal $[\mathfrak{a}]$ is the secret key.

The vertices in the vOW graph that we are considering here are elliptic curves that lie a specific size subgroup action away from $E_0$ or $E_1$ There are various ways to decide on the size of this group action. In this analysis, we consider two different split-ups when the connecting ideal between $E_0$ and $E_1$ is of the shape $[\mathfrak{l}_1]^{e_1}[\mathfrak{l}_2]^{e_2}\cdots[\mathfrak{l}_n]^{e_n}$.

1. Introduced in the original CSIDH paper [9], split up our prime ideals into two disjoint sets such that the sum of the norms of the ideals in each set is as close together as possible and let their exponents $e_i \in [-B, B]$, for $B$ a bound on the exponents.
2. Use ideals of the form $[\mathfrak{l}_1]^{e_1}[\mathfrak{l}_2]^{e_2}\cdots[\mathfrak{l}_n]^{e_n}$ for $e_n \in [-a, a]$ for some integer $a \leq B$ for both sides.

No matter the choice of the split, the attack continues in the same way. The starting point of our trails is simply created by starting at $E_0$ or $E_1$ at random and then using a random CSIDH key to a random elliptic curve (i.e., $[\mathfrak{l}_1]^{e_1}[\mathfrak{l}_2]^{e_2}\cdots[\mathfrak{l}_n]^{e_n}$ for $e_n \in [-B, B]$). Due to the randomization properties of ideals of this form, on which CSIDH is based, we can assume we have a random starting position for our trail.

To define $f$ we need the following notation.

- $\mathcal{V}_s$ denotes the set of all possible ideals of the chosen shape for the side $s \in \{0, 1\}$.
- $I_s = \{1, ..., n\}$ a set with $n = |\mathcal{V}_s|$.

– $S = \{0,1\} \times I$ is how we define the domain and codomain of $f$.

We define a function $f : S \to S$ by using the following functions.

1. $g_s : I_s \to \mathcal{V}_s$, which takes the input for $I_s$ and uses it to create the vector of exponents for that specific side $s$.
2. $\psi_s : \mathcal{V}_s \to \mathbb{F}_p$, which takes a group action on $E_s$ induced by the element of $\mathcal{V}_s$ and returns the Montgomery coefficient of the codomain curve.
3. $H : \mathbb{F}_p \to \{0,1\}^*$, which hashes the found Montgomery coefficient and outputs a pseudorandom integer.
4. $w : \{0,1\}^* \to S$, which takes a random bit-input and outputs a side $s$ and a random vector exponent from $I_s$.

Then $f$ is defined as:

$$f(s,i) = w \circ H \circ \psi_s \circ g_s(i).$$

We again have the same issues as in the $\mathbb{F}_{p^2}$ case, i.e., a collision is not necessarily a golden collision since the collision might be between two elements induced by the same side[6] or a hash collision can occur.

**A discrete logarithm variant.** We now introduce a discrete logarithm variant for the $\mathbb{F}_p$ case. To this end, we ignore the additional assumption of keys being low-exponent vectors, so that we are able to walk through the entire search space without restrictions. Using the same considerations as in the $\mathbb{F}_{p^2}$ case, it is even more natural to define this random walk as an analogue of the discrete logarithm Pollard's walk. Our walk is over the whole $\mathbb{F}_p$ graph where the vertices are the elliptic curves defined over $\mathbb{F}_p$ and the edges are the possible $\ell_i$-isogenies for $i \in \{1, ..., n\}$. Since all elliptic curves defined over $\mathbb{F}_p$ can be written in Montgomery form, we will use the Montgomery coefficient to denote the curve we are at.

A starting point is computed by computing a full CSIDH key applied to whichever side we are starting the trail from. Thus, we ensure that we end up on a random point. A trail is then created starting from this point by deterministically taking $[\mathfrak{l}]$-actions based on the Mongomery coefficient of the current elliptic curve. When $p = 4\Pi_{i=1}^{n}\ell_i - 1$, we can apply group actions of the form $\mathfrak{l}_i = (\ell_i, \pi - 1)$ for $i \in \{1, ..., n\}$. We denote by $E_A$ the curve $y^2 = x^3 + Ax^2 + x$ with Montgomery coefficient $A$, and $M(E)$ the Montgomery coefficient of the curve $E$. The pseudo-random function $f : \mathbb{F}_p \to \mathbb{F}_p$ is defined as follows

$$f(A) = \begin{cases} M([\mathfrak{l}_1] * E_A) & \text{if } A \equiv 0 \mod n \\ M([\mathfrak{l}_2] * E_A) & \text{if } A \equiv 1 \mod n \\ \quad \vdots & \\ M([\mathfrak{l}_n] * E_A) & \text{if } A \equiv n-1 \mod n. \end{cases}$$

[6] In contrast to the $\mathbb{F}_{p^2}$ setting, here, endomorphisms obtained in this way are useless.

Since in this case an endomorphism is not useful, a collision only counts if the collision is between two trails starting at differing starting curves. If $n$ is large and smooth enough, i.e., we have enough primes to choose from, this is a function that behaves randomly. Points are easily distinguishable by some property of the Montgomery coefficient, for example, the number of trailing zeros.

A useful property of the action on the $\mathbb{F}_p$ isogeny graph is that the twist of $E_0$ is itself. This means that if we manage to find a path between $E_0$ and the twist of $E_1$, by for example finding a collision between a trail from $E_0$ to $E'$ and a trail from the twist of $E'$ to $E_1$. We use this to reduce the runtime by, instead of looking at a Montgomery coefficient, looking at its absolute value (modulo $p$). This creates an equivalence relation between the $A$ coefficients of the curve and in practice, we take the minimum of the two to be the representative. When a distinguished point is found, the representative is computed and stored. This implies that we halve the search space by looking at the curve and the twist at the same time. We calculate the birthday bound in the usual way, but the original $N$ now is halved. Hence, we obtain $\mathcal{O}(\sqrt{\pi N/2})$ for the complexity, which is a speedup of a factor of $\sqrt{2}$. This is in contrast with the result of applying this optimization to the meet-in-the-middle attack, where the speedup factor is 2 [9].

**Comparison of the attacks.** Now let us compare the three attacks that we have explained here, which are:

1. The meet-in-the-middle variant, where we split up the ideals into two disjoint sets. Concretely, we split up at the integer $\nu$ such that $\sum_{k=1}^{\nu} \mathfrak{l}_k \approx \sum_{k=\nu+1}^{n} \mathfrak{l}_k$.
2. The meet-in-the-middle variant, where both sides use $[\mathfrak{l}_1]^{e_1}[\mathfrak{l}_2]^{e_2} \cdots [\mathfrak{l}_n]^{e_n}$ for $e_n \in [-a, a]$ for varying $a$.
3. The proposed discrete logarithm variant.

We will simply denote them as Attack 1, 2 and 3. We first show the toy example primes we are using in Table 3, where we also show the split of the factors used in Attack 1.

**Table 3.** Example primes used in further experiments, including the split used in Attack 1 (only for the smallest prime since the others take too long).

| Prime | # bits | Factors | Split |
|---|---|---|---|
| 117 010 739 | 27 | $[3, 5, 7, 11, 19, 31, 43]$ | $[3, 5, 7, 11, 31], [19, 43]$ |
| 8 342 459 579 | 33 | $[3, 5, 7, 13, 19, 29, 47, 59]$ | - |
| 442 150 357 739 | 39 | $[3, 5, 7, 13, 19, 29, 47, 53, 59]$ | - |
| 92 879 716 861 931 | 47 | $[3, 7, 11, 13, 19, 29, 31, 41, 61, 181]$ | - |

Now for the comparison on how many $f$-evaluations are needed to find a (golden) collision, see Table 4. Important to note here is that this table counts

$f$-evaluations, which do not cost the same in different attacks, with Attack 3 being disadvantaged in such a comparison. But even ignoring the exact cost, we can already conclude that Attack 3 is the cheapest. Now if we also take into account the average cost of one $f$-evaluation, the difference becomes even greater. Table 5 shows the average cost we expect for each of the attacks, assuming that since we are working with small primes $\ell_i$ we can compute the isogeny linear in the sum of the prime degree-factors using Vélu's formulas [37]. For Attack 1, one $f$-evalution consists in computing an isogeny of degree $d \in \{0, \dots, B\}$ for half of the possible primes, say an average of 2.5 $\ell_i$-isogeny for the original CSIDH parameters. So this will get us an average cost of $\sum_{i=1}^{n} \ell_i \cdot 5/4$, and the cost for Attack 2 is calculated in the same way but with the chosen bound and with taking all the primes for each evaluation. For Attack 3 however, we only need to do one $\ell_i$-isogeny per $f$-evalution, so we average over the sum of all primes $\sum_{i=1}^{n} \ell_i/n$.

**Table 4.** Comparison of different meet-in-the-middle variant (Attack 1,2 and 3) and the discrete logarithm variant attack (Attack 4) in the $\mathbb{F}_p$ graph (average of 1000 runs). Cost is number of applications of the pseudorandom $f$-function. Dashes mean these attacks are infeasible to run for these parameters in our Sagemath version.

| Number of bits | 27 | 33 | 39 | 47 |
|---|---|---|---|---|
| Attack 1 | 11 360.5 | - | - | - |
| Attack 2, $a = 2$ | 804.2 | 1 592.4 | 2 966.4 | - |
| Attack 2, $a = 3$ | 787.1 | 1 584.3 | 2 783.3 | - |
| Attack 2, $a = 4$ | 770.5 | 1 632.4 | 2 883.372 | - |
| Attack 2, $a = 5$ | 785.5 | 1 545.4 | 2 759.8 | - |
| Attack 3 | 27.3 | 49.0 | 135.0 | 2476.0 |

**Table 5.** Comparison of the average cost of $f$ in the different attacks in the $\mathbb{F}_p$ graph.

| | Attack 1 | Attack 2 | Attack 3 |
|---|---|---|---|
| Average cost | $O(\sum_{i=1}^{n} \ell_i \cdot 5/4)$ | $O(\sum_{i=1}^{n} \ell_i \cdot a/2)$ | $O(\sum_{i=1}^{n} \ell_i/n)$ |

## 4 Collision search for the multi-user problems

In this section, we describe how we can attack the multi-user problems defined in Section 2.3 through construction of multi-user graphs. The vertices of the graph always correspond to public keys of users and the edges correspond to known (non-trivial) relations between them found by collision attacks. From these relations, when we know the secret key of one of the users, we can compute

the secret key of some of the other users as well. Thus, in the multi-user graph, the knowledge of the secret key of one user (an infiltrator) propagates to all users in its connected component. We show that we can remove the need for an infiltrator in the algorithm when cycles in the graph can be used to recover secret keys.

**The general setup.** Our treatment of multi-user graphs as directed graphs might be counterintuitive, since the edges represent *relations* between the public keys. The reason we make this editorial choice is that, in our multi-user algorithms, we are aiming to avoid *useless* collisions. We consider a collision to be useless if it does not result in recovering more secret keys than we can recover from previously obtained collisions. Hence, a collision is useless if both endpoints belong to a connected component that already contains a cycle or an infiltrator. In other words, let $U$ be the dynamic set of users whose secret key is not yet recovered, initialized at the start as $U = \{u_1, \ldots, u_L\}$. A collision is then useful if at least one of the endpoints of the edge it produces is in $U$. Hence, to avoid useless collisions, all processes need to produce distinguished points belonging to users in this set. Note that a distinguished point *belonging* to a user $u_i$ refers to the starting point of its trail. For example, in the discrete logarithm case, trails with a starting point $R = aP + bQ_i$, where $Q_i$ is the public key of user $u_i$, belong to user $u_i$. We can thus have an algorithmic solution to the problem of useless collisions: we only produce distinguished points starting from points connected to users in $U$, and we update the set $U$ every time we find a collision by removing (only!) one of the users involved in the collision. A crucial point to note here is that this is only possible if only the starting point is user-dependent but not the random walk function. Having these user-independent walks allows for use of precomputation for the discrete logarithm problem. In our case, the consequence of a walk being user-dependent is that we would not be able to remove elements from $U$, since that would require modifying the random walk function. Recall that a crucial requirement for the vOW method is to have a *deterministic* random walk. If we modify the random walk function dynamically, the entire vOW analysis breaks down since we no longer have the core property that two trails that collide on a non-distinguished point will continue walking on the same path and eventually arrive at a distinguished point and discover the collision. When this behavior breaks, we can no longer rely on the birthday paradox for the analysis of the expected number of steps.

We start with a graph with an empty edge set. We then add edges by finding relations using the collision-search methods described in Section 3 in parallel. Each thread is tasked with finding a relation involving a specific user $u_i$. This means that the trails that this thread produces have a starting point that is dependent on user $u_i$. When the thread finds a collision with a trail computed by user $u_j$, we recover a relation between user $u_i$ and user $u_j$ and obtain a new edge in the graph. Note that the case $i = j$ is also allowed. The algorithm ends when the desired graph configuration is obtained. This graph of relations is then

used to recover the secret keys. For the multi-user algorithms, we analyse which of the following properties are achieved in the different problem settings.

- We can avoid producing useless collisions.
- There is no need for an infiltrator in the algorithm.
- We recover the secret key of *all* users (deterministically).

We always achieve the first outcome as described above. The two remaining outcomes are intrinsically related and depend on whether secret keys can be computed from a cycle. A summary of the results can be found in Table 1. Before we go into the explanation of these results we will analyze the created multi-user graph in a bit more detail.

The first property also allows us to have a predictable count on the number of collisions required for the attack. The complexity of the overall algorithm then directly follows from the following lemma.

**Lemma 5.** *[34, Theorem 2] The expected number of steps of a random walk in a graph of $L$ vertices using the $f$ mapping defined in Problem 7 before $k$ collisions occur is asymptotic to*

$$\sqrt{2kL}.$$

The multi-user graph that we create is a functional graph. These graphs have the property that each connected component has a cycle, and they have exactly as many vertices as edges. On the other hand, each graph that has a cycle in each connected component has at least as many edges as vertices. We deduce that functional graphs are optimal for our goal.

We can solve the multi-user problems in two different ways.

- In some cases a cycle (or multiple cycles) allows us to solve the problem for all users in the cycle, and by inference for all users connected to a cycle. This way we solve the problem because the constructed multi-user graph is a functional graph.
- When a cycle does not give enough information, we slightly modify the construction by including a fictive user $u_0$ for which we know the secret key. This is the user we refer to as an infiltrator. This way we partially solve the related problem, in fact we recover the secret key of all the users in the same connected component as $u_0$.

We will treat in more detail each of the three multi-user problems. We will always denote by $L$ the number of users and by $N$ the cardinality of the set we act on. First we analyze the multi-user graph created in this way.

**Analysis of the multi-user graph.** To be able to estimate the size of the largest component we have to take a closer look at how the multi-user graph is created. One might think it is a simple uniform functional graph where each vertex has exactly one outgoing edge and those edges connect uniformly to one of

the other vertices. But due to the algorithmic choices we make to avoid producing useless collisions, this is not what happens in practice. In practice, users that get connected early in the process have a smaller probability to get connected to. This happens since we stop computing trails for a user that has already discovered a collision. This means that this user gets less prevalent in our gathered distinguished points over time. Hence, the placement of the endpoints of edges does not necessarily follow a uniform distribution. We can however calculate the likelihood of some events, that will let us infer some properties that we are interested in.

Let us adopt a labeling of the vertices that indicates the order in which they receive an outgoing edge. The $i$th vertex is thus the vertex that received an outgoing edge with the $i$th collision in the multi-threaded vOW search. Following this labeling, when an outgoing edge from vertex $m$ is added (thus at the $m$th collision), we can calculate the probability of its endpoint being a given vertex $k$. We use the result from Lemma 5 that states we need $\sqrt{2Lm}$ trails to find $m$ collisions. So between the $(m-1)$th and $m$th collision we gain $\theta\sqrt{2Lm} - \theta\sqrt{2L(m-1)}$ trails, but these only start at vertices that are still not attached (which is $L-m$). So we can define the probability of vertex $k$ being an endpoint at the time of collision $m$ by simply counting the expected number of trails created for that specific vertex $k$ and dividing it by the total number of trails collected up to that point, that is

$$p(m,k) = \begin{cases} \frac{\sum_{i=1}^{k}\gamma(i)}{\sqrt{2m}} & \text{for } k \leq m, \\ \\ \frac{\sum_{i=1}^{m}\gamma(i)}{\sqrt{2m}} & \text{for } k > m, \end{cases}$$

where $\gamma(j) = (\sqrt{2j} - \sqrt{2(j-1)})/(L-j+1)$. Note that the $\sqrt{L}$ gets divided out.

Now that we have these probabilities, we can compute the average weakly connected degree (the degree of a vertex neglecting the edge direction) that we expect for each vertex $u$ as

$$\mathbb{E}(\deg(u)) = 1 + \sum_{i=1}^{L} p(i,u). \tag{3}$$

We now use the Molloy-Reed criterion [28] which states that[7] if the ratio

$$\frac{\sum_{i=1}^{L}\mathbb{E}(\deg(i))^2}{\sum_{i=1}^{L}\mathbb{E}(\deg(i))}$$

is larger than 2, then we can expect to have an indentifiable giant component. Note that, for small $L$ this starts below the bound of 2, but already around $n = 30$ it passes this bound.

[7] Definition adapted for our purposes.

Now that we have shown the existence of a giant component in our multi-user graph, we are interested in the size of this giant component. If our graph was a uniform functional graph we would know that this size is about 76%, as shown in Lemma 2. However, since we do not have a perfectly uniform graph, let us look at our graph as an Erdős-Rényi graph [18], which is a graph defined by a number of vertices and an associated probability for each edge of being present or absent. In this model, the expected fraction of vertices $S$ in the giant component satisfies $S = 1 - e^{-cS}$, where $c$ is the average weakly connected degree of the vertices. For $c = 1.98$, which matches our case as per Equation (3), one can numerically approximate that $S = 0.79$.

We conjecture thus that our graph will have a giant component around this interval of 76 to 79%. Small scale experiments have shown that we usually get about 75% for graphs of about a thousand vertices and the percentage does slowly increase when increasing the size of the graph. We thus assume that we will find a large component of approximately 75% as a cautious choice.

Similar to the uniform case, one needs an average degree of 4 to get the giant component to encompass 98% of the graph, which would lead to needing $2L$ collisions.

Knowing the number of weakly-connected components in the case of this non-uniform functional graph would be very nice to know, but this is not an easy analysis and is left to future work.

### 4.1 Discrete log

The random walk function used in the (elliptic curve) discrete logarithm attack can easily be defined as user-independent, for instance as

$$f(R) = \begin{cases} R + r_1 P & \text{if } R \in S_1 \\ R + r_2 P & \text{if } R \in S_2 \\ R + r_3 P & \text{if } R \in S_3. \end{cases} \tag{4}$$

for $r_1, r_2, r_3$ integers and $\{S_1, S_2, S_3\}$ are pairwise disjoint and form a partition of the group $G$. The graph is built as described in [20], that is, the vertices are the public keys $Q_i$ of the users and the edges are linear relations between their secret keys. A slight difference is that since we are able to avoid generating useless edges, we obtain a functional graph. A collision between two public keys $Q_k = x_k P$ and $Q_l = x_l P$ represents the equality

$$aP + bQ_k = cP + dQ_l. \tag{5}$$

for some known values $a, b, c, d$. This yields the linear relation

$$a + bx_k - c - dx_l = 0. \tag{6}$$

A cycle $\{v_1, \ldots, v_n\}$ yields $n$ collisions, resulting in the following linear system of equations modulo $N$, the order of $P$

$$\begin{cases} a_1 + b_1 x_1 - c_1 - d_1 x_2 = 0 \\ a_2 + b_2 x_2 - c_2 - d_2 x_3 = 0 \\ \vdots \\ a_{n-1} + b_{n-1} x_{n-1} - c_{n-1} - d_{n-1} x_n = 0 \\ a_n + b_n x_n - c_n - d_n x_1 = 0. \end{cases}$$

Suppose that $N$ is a prime number $p$. Since all the $b_i$ are not zero by construction, the first $n-1$ equations are linearly independent because for any given subset of these equations there is at least one variable $x_i$ which appears in exactly one equation. Hence the rank of the linear system is at least $n-1$. Moreover the system has a unique solution $(x_1, \ldots, x_n) \in \mathbb{F}_p^n$ (i.e. the rank is $n$) with high probability. In fact, the first equation gives $x_2$ as a linear function of $x_1$ which can be substituted in the second one giving $x_3$ as a linear function of $x_1$ and so on until $x_n$ is equal to a linear function of $x_1$ from the second to last relation. Substitution into the last equation yields an equation in $x_1$ of degree 1, or possibly an underdetermined equation. The system can be underdetermined and this happens if and only if the last relation is implied by the previous ones.

According to Lemma 3, after $L$ (useful) collisions for $L$ users, we have built a functional graph where each connected component has a cycle. For each component, we first solve a linear system to recover the discrete logs of all users in the cycle, then we use linear relations as in Equation (6) to recover the discrete logarithm of all remaining users. A cycle in the graph can be a loop, that is, an edge that connects a vertex to itself, as this allows us to recover the secret key of the corresponding vertex and thus all the secret keys in the connected component. We exclude trivial relations, and in this case, the only trivial relations are those as in Equation (5) where additionally we have $b_i = b_j$ and $k = l$ (forcing also $a_i = a_j$). This case has negligible occurrence probability and it is also excluded in the one-collision discrete logarithm case. The complexity of the algorithm is $\mathcal{O}(\sqrt{2LN})$ and all three desired properties are achieved.

### 4.2 The $\mathbb{F}_{p^2}$ graph

When solving Problem 5, we define the multi-user graph as follows. The vertices are again the public keys of users, so they are isomorphism classes of supersingular elliptic curves denoted by their $j$-invariant. The edges are recovered $\ell^e$-isogenies between them, for some variable $e$.

Even though in the $\mathbb{F}_{p^2}$ case cycles are also useful, similar to the discrete logarithm case, a cycle does not immediately give us a secret key recovery. Thus we start by looking at the attack as done in the original multi-user attack on the discrete logarithm by [20] and introduce an infiltrator. Then we will introduce a way to exploit the cycles, but conclude that this is not the most efficient solution in practice.

**Approach 1: introducing the infiltrator.** In this case, an infiltrator is the curve $E_0$, both because this is a curve with known endomorphism ring and because the problem statement asks each user curve to find an isogeny to $E_0$. Since the knowledge of the endomorphism ring can be pushed through isogenies, any curve with a known isogeny to $E_0$ can be an infiltrator, and we choose $E_0$ itself for this exposition. We then compute one collision for each user. Since we expect to get a weakly-connected giant component containing about 75% of the vertices, we have about a 3/4 probability that $E_0$ will be in this giant component. In that case, we break all keys in the giant component, since we know an $\ell^e$-isogeny from $E_0$ to each of these public keys. Do note that most of these isogenies are a composition of many isogenies each of degree about $p$, thus quite large.

Putting $N \sim p/12$, that is, the number of supersingular elliptic curves over $\mathbb{F}_{p^2}$, this attack has complexity $\mathcal{O}(\sqrt{LN})$ times the cost of computing one step of the iterated function, which is $\mathcal{O}(\log p)$.

Generally speaking, if an algorithm $\mathcal{A}$ partially solves Problem 5 with complexity $\mathcal{O}(C_N\sqrt{L})$ for some constant $C_N$ independent of $L$ (in this case, $C_N = \sqrt{N}$), and recovers a fraction $\alpha$ of the keys with probability $p$, then there is an algorithm that calls $\mathcal{A}$ repeatedly and solves (fully) Problem 5 with average complexity $\mathcal{O}\left(\frac{C_N\sqrt{L}}{p(1-\sqrt{1-\alpha})}\right)$. In fact, we iterate $\mathcal{A}$ on average $1/p$ times to get a partial solution, at which point we remain with $(1-\alpha)L$ unsolved instances. We run $\mathcal{A}$ on the unsolved instances, after which $(1-\alpha)^2 L$ instances are still unsolved. We continue until all keys are recovered. The total complexity is asymptotic to

$$\frac{1}{p} C_N \sum_{n=0}^{\infty} \sqrt{(1-\alpha)^n L} = \frac{C_N\sqrt{L}}{p(1-\sqrt{1-\alpha})}.$$

Hence the described algorithm that partially solves Problem 5 also gives an algorithm for solving Problem 5 with complexity $\frac{1}{0.75(1-\sqrt{1-0.75})} \sim 2.67$ times larger.

**Approach 2: leveraging the cycles.** Cycles in the multi-user graph induce endomorphisms: since a collision found between $E_1$ and $E_2$ ensures an $\ell^e$-isogeny from $E_1$ to $E_2$ for some $e$, a cycle in the collision graph generates $\ell^e$-endomorphisms for some integer $e$ for all the elliptic curves in the given component. Given a cycle $\{E_1, \ldots, E_n\}$ we build an endomorphism of $E_1$ as the composition $\phi_n \circ \phi_{n-1} \circ \cdots \circ \phi_1$ where $\phi_i$ denotes the isogeny from $E_i$ to $E_{i+1}$ of degree a power of $\ell$ obtained from the collision. Note that a cycle can also be a simple self-loop, which is useful as long as it is not a scalar endomorphism (the occurrence of which is negligible compared to the favorable case).

Now contrary to the discrete logarithm case, one cycle does not immediately give us all the keys in a connected component. Even though a cycle will most likely induce a non-trivial endomorphism $\alpha$, the non-quantum complexity of $|\mathrm{disc}(\mathbb{Z}[\alpha])|^{1/4}$ of recovering the endomorphism ring given $\alpha$ [27] is tightly connected to the size of its degree. Since the endomorphisms we obtain from

cycles are humongous[8], we are not able to apply the result with any advantage over the infiltrator approach.

The conclusion is similar for the approach where we need two distinct suitable independent endomorphisms to find the entire ring. In this case, we first construct the multi-user graph as a functional graph by computing one collision for each user. We can then add a small amount of edges (about $1/2 \log(0.25 \cdot L)$) such that each connected component has at least two independent cycles. We prove the relevant graph-theoretic lemma in Section A. It might happen that the two induced endomorphisms are not independent, however, this event has low probability (see Section B). In this unlikely case we can keep adding edges until two independent endomorphisms are found. The approach (detailed further in Section B) however contains a factoring step which can be solved in only subexponential non-quantum time.

One can also consider a hybrid approach where the keys in the largest component are found via the infiltrator and the remaining keys are recovered through the use of cycles (now expected to induce smaller endomorphisms, being of length at most $0.25L$ when the largest component is excluded). However, the main limit of any approach using cycles is the non-existance of a polynomial time algorithm that given two endomorphisms computes the endomorphism ring. Approach 1 is thus deemed optimal in practice and presented in Table 1.

### 4.3 The $\mathbb{F}_p$ graph

In the $\mathbb{F}_p$ graph, a collision in our multi-user graph is simply a class group element $k$ that maps $E_1$ to $E_2$ using the isogeny kernel representation where the kernel is composed of small $\mathbb{F}_p$-rational subgroups. Hence, the multi-user graph again has vertices corresponding to elliptic curves, this time represented with the Montgomery coefficient, and edges represent smooth-degree $\mathbb{F}_p$-rational isogenies between them. A cycle is then an $\mathbb{F}_p$-rational endomorphism. This endomorphism is thus already contained in the quadratic order and can not help us recover the full endomorphism ring of the curve. It simply gives us another equivalence relation, which is only a small step towards recovering the ideal class group structure. This means that we can not use cycles, and so we do need an infiltrator. The role of the infiltrator is again played by the $E_0$ curve.

We run the multi-user algorithm in the usual way, requiring one collision for each user and thus obtaining a functional graph. The starting points of trails are defined similarly as in the $\mathbb{F}_{p^2}$ case. We compute an arbitrary isogeny of degree approximately $\sqrt{p}$ starting from curve $E_i$ and take the image curve as a starting point. From the obtained functional graph, we can recover all of the secret keys of the connected component that contains the infiltrator. Statistically, we would expect this infiltrator to be in the largest connected component, as that one is expected to be of size $0.76L$, as per Lemma 2. Then we can easily recover all secret keys of this component, but none of the others. Since the class group elements can be easily combined by simply adding up the exponent

[8] They are of degree approximately $p^k$ for a cycle of length $k$.

vector, it is easy to find isogenies from our starting curve $E_0$ to each of the curves in the largest connected component. Note that adding those keys will give us large keys, which will make them less efficiently computable, and also out of the predefined CSIDH keyspace. We can apply some key reduction techniques. The first one is straightforward, it consists in removing exponents that cancel out. We can also use the known class group relations, in addition to those we obtain from cycles (in any of the connected components), to reduce the $L_1$-norm of the exponent vectors. The known class group relations we refer to, proven independently in both [10, Lemma 8] and [29], tell us that the exponent vector $(3, \ldots, 3)$ corresponds to the identity action and that $(1, \ldots, 1)$ can be computed alternatively by computing a negative 4-isogeny. These techniques can not help us get small enough exponents, but for cryptanalysis purposes, we do not require the group action computation to be as efficient. For the CSIDH-512 parameter set, the lattice $L$ has been explicitly computed allowing for fast action evaluation of any vector.

The analysis and results we obtain in this case follow exactly the ones in [20]. The algorithm as described runs with complexity $\mathcal{O}(\sqrt{LN})$, as we require one collision per user, and it recovers the secret key of about 76% of users. There is a natural trade-off between the running time and the proportion of recovered keys, since the expected size of the largest connected component grows with the number of edges. The use of multiple infiltrators is also possible, but this simply corresponds to giving more computing power to the thread that is working for user 0.

**Acknowledgements.**

We thank Sorina Ionica and Gilles Dequen for initial discussions on the discrete logarithm case and the use of cycles.
The work was supported by the Dutch Ministry of Education, Culture, and Science through Gravitation project "Challenges in Cyber Security - 024.006.037" and by the Dutch Research Council (NWO) through project 613.009.144.

## References

[1] Gora Adj, Daniel Cervantes-Vázquez, Jesús-Javier Chi-Domínguez, Alfred Menezes, and Francisco Rodríguez-Henríquez. On the cost of computing isogenies between supersingular elliptic curves. In Carlos Cid and Michael J.: Jacobson, Jr., editors, *SAC 2018: 25th Annual International Workshop on Selected Areas in Cryptography*, volume 11349 of *Lecture Notes in Computer Science*, pages 322–343, Calgary, AB, Canada, August 15–17, 2019. Springer, Cham, Switzerland.

[2] Navid Alamati, Luca De Feo, Hart Montgomery, and Sikhar Patranabis. Cryptographic group actions and applications. In Shiho Moriai and Huaxiong Wang, editors, *Advances in Cryptology – ASIACRYPT 2020, Part II*, volume 12492 of *Lecture Notes in Computer Science*, pages 411–439, Daejeon, South Korea, December 7–11, 2020. Springer, Cham, Switzerland.

[3] Yves Aubry, Roger Oyono, and Christelle Vincent. Minimal degree of an isogeny between a supersingular elliptic curve and its conjugate, 2026.

[4] Gregory Berkolaiko and Peter Kuchment. *Introduction to Quantum Graphs*, volume 186 of *Mathematical Surveys and Monographs*. American Mathematical Society, Providence, RI, 2013.

[5] Daniel J. Bernstein and Tanja Lange. Computing small discrete logarithms faster. In Steven D. Galbraith and Mridul Nandi, editors, *Progress in Cryptology - INDOCRYPT 2012: 13th International Conference in Cryptology in India*, volume 7668 of *Lecture Notes in Computer Science*, pages 317–338, Kolkata, India, December 9–12, 2012. Springer Berlin Heidelberg, Germany.

[6] Daniel J. Bernstein and Tanja Lange. Non-uniform cracks in the concrete: The power of free precomputation. In Kazue Sako and Palash Sarkar, editors, *Advances in Cryptology – ASIACRYPT 2013, Part II*, volume 8270 of *Lecture Notes in Computer Science*, pages 321–340, Bengalore, India, December 1–5, 2013. Springer Berlin Heidelberg, Germany.

[7] 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, Part I*, volume 11921 of *Lecture Notes in Computer Science*, pages 227–247, Kobe, Japan, December 8–12, 2019. Springer, Cham, Switzerland.

[8] I. R. Borevich, Z. I.; Shafarevich. Number theory. 1986.

[9] Wouter Castryck, Tanja Lange, Chloe Martindale, Lorenz Panny, and Joost Renes. CSIDH: An efficient post-quantum commutative group action. In Thomas Peyrin and Steven Galbraith, editors, *Advances in Cryptology – ASIACRYPT 2018, Part III*, volume 11274 of *Lecture Notes in Computer Science*, pages 395–427, Brisbane, Queensland, Australia, December 2–6, 2018. Springer, Cham, Switzerland.

[10] Wouter Castryck, Lorenz Panny, and Frederik Vercauteren. Rational isogenies from irrational endomorphisms. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology – EUROCRYPT 2020, Part II*, volume 12106 of *Lecture Notes in Computer Science*, pages 523–548, Zagreb, Croatia, May 10–14, 2020. Springer, Cham, Switzerland.

[11] Denis Xavier Charles, Kristin E. Lauter, and Eyal Z. Goren. Cryptographic hash functions from expander graphs. *Journal of Cryptology*, 22(1):93–113, January 2009.

[12] Jesús-Javier Chi-Domínguez, Andre Esser, Sabrina Kunzweiler, and Alexander May. Low memory attacks on small key CSIDH. In Mehdi Tibouchi and Xiaofeng Wang, editors, *ACNS 2023: 21st International Conference on Applied Cryptography and Network Security, Part II*, volume 13906 of *Lecture Notes in Computer Science*, pages 276–304, Kyoto, Japan, June 19–22, 2023. Springer, Cham, Switzerland.

[13] Pierrick Dartois, Jonathan Komada Eriksen, Tako Boris Fouotsa, Arthur Herlédan Le Merdy, Riccardo Invernizzi, Damien Robert, Ryan Rueger, Frederik Vercauteren, and Benjamin Wesolowski. PEGASIS: Practical effective class group action using 4-dimensional isogenies. In Yael Tauman Kalai

and Seny F. Kamara, editors, *Advances in Cryptology – CRYPTO 2025, Part I*, volume 16000 of *Lecture Notes in Computer Science*, pages 67–99, Santa Barbara, CA, USA, August 17–21, 2025. Springer, Cham, Switzerland.

[14] Luca De Feo. Mathematics of isogeny based cryptography. In *Post-Quantum Cryptography*, volume 10346 of *Lecture Notes in Computer Science*, pages 3–30. Springer, 2017.

[15] 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, Part III*, volume 11478 of *Lecture Notes in Computer Science*, pages 759–789, Darmstadt, Germany, May 19–23, 2019. Springer, Cham, Switzerland.

[16] Luca De Feo, David Kohel, Antonin Leroux, Christophe Petit, and Benjamin Wesolowski. SQISign: Compact post-quantum signatures from quaternions and isogenies. In Shiho Moriai and Huaxiong Wang, editors, *Advances in Cryptology – ASIACRYPT 2020, Part I*, volume 12491 of *Lecture Notes in Computer Science*, pages 64–93, Daejeon, South Korea, December 7–11, 2020. Springer, Cham, Switzerland.

[17] Christina Delfs and Steven D. Galbraith. Computing isogenies between supersingular elliptic curves over $\mathbb{F}_p$. *Designs, Codes and Cryptography*, 78(2):425–440, 2016.

[18] P. Erdös and A. Rényi. On random graphs i. *Publicationes Mathematicae Debrecen*, 6:290, 1959.

[19] Philippe Flajolet and Andrew M. Odlyzko. Random mapping statistics. In Jean-Jacques Quisquater and Joos Vandewalle, editors, *Advances in Cryptology – EUROCRYPT'89*, volume 434 of *Lecture Notes in Computer Science*, pages 329–354, Houthalen, Belgium, April 10–13, 1990. Springer Berlin Heidelberg, Germany.

[20] Pierre-Alain Fouque, Antoine Joux, and Chrysanthi Mavromati. Multi-user collisions: Applications to discrete logarithm, Even-Mansour and PRINCE. In Palash Sarkar and Tetsu Iwata, editors, *Advances in Cryptology – ASIACRYPT 2014, Part I*, volume 8873 of *Lecture Notes in Computer Science*, pages 420–438, Kaoshiung, Taiwan, R.O.C., December 7–11, 2014. Springer Berlin Heidelberg, Germany.

[21] Steven D. Galbraith. Constructing isogenies between elliptic curves over finite fields. *LMS Journal of Computation and Mathematics*, 2:118–138, 1999.

[22] David Jao and Luca De Feo. Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies. In Bo-Yin Yang, editor, *Post-Quantum Cryptography - 4th International Workshop, PQCrypto 2011*, pages 19–34, Tapei, Taiwan, November 29 – December 2 2011. Springer Berlin Heidelberg, Germany.

[23] David R. Kohel. Endomorphism rings of elliptic curves over finite fields. 1996.

[24] Martin D. Kruskal. The expected number of components under a random mapping function. *American Mathematical Monthly*, 61(6):392–397, 1954.

[25] Fabian Kuhn and René Struik. Random walks revisited: Extensions of Pollard's rho algorithm for computing multiple discrete logarithms. In Serge Vaudenay and Amr M. Youssef, editors, *SAC 2001: 8th Annual International Workshop on Selected Areas in Cryptography*, volume 2259 of *Lecture Notes in Computer Science*, pages 212–229, Toronto, Ontario, Canada, August 16–17, 2001. Springer Berlin Heidelberg, Germany.

[26] Alexander May and Massimo Ostuzzi. Multiple group action dlogs with(out) precomputation. In Tibor Jager and Jiaxin Pan, editors, *PKC 2025: 28th International Conference on Theory and Practice of Public Key Cryptography, Part III*, volume 15676 of *Lecture Notes in Computer Science*, pages 364–387, Røros, Norway, May 12–15, 2025. Springer, Cham, Switzerland.

[27] Arthur Herlédan Le Merdy and Benjamin Wesolowski. The supersingular endomorphism ring problem given one endomorphism. *IACR Communications in Cryptology (CiC)*, 2(1):6, 2025.

[28] M. Molloy and B. Reed. A critical point for random graphs with a given degree sequence. In *Proceedings of the Sixth International Seminar on Random Graphs and Probabilistic Methods in Combinatorics and Computer Science, "Random Graphs '93" (Poznań, 1993)*, volume 6, pages 161–179, 1995.

[29] Hiroshi Onuki and Tsuyoshi Takagi. On collisions related to an ideal class of order 3 in CSIDH. In Kazumaro Aoki and Akira Kanaoka, editors, *IWSEC 20: 15th International Workshop on Security, Advances in Information and Computer Security*, volume 12231 of *Lecture Notes in Computer Science*, pages 131–148, Fukui, Japan, September 2–4, 2020. Springer, Cham, Switzerland.

[30] Aurel Page and Benjamin Wesolowski. The supersingular endomorphism ring and one endomorphism problems are equivalent. In Marc Joye and Gregor Leander, editors, *Advances in Cryptology – EUROCRYPT 2024, Part VI*, volume 14656 of *Lecture Notes in Computer Science*, pages 388–417, Zurich, Switzerland, May 26–30, 2024. Springer, Cham, Switzerland.

[31] John M. Pollard. Monte carlo methods for index computation ( (mod $p$)). *Mathematics of Computation*, 32(143):918–924, 1978.

[32] Joseph H. Silverman. *The Arithmetic of Elliptic Curves*, volume 106 of *Graduate Texts in Mathematics*. Springer, New York, 2 edition, 2009.

[33] Edlyn Teske. On random walks for pollard's rho method. *Mathematics of Computation*, 70(234):809–825, 2001.

[34] Monika Trimoska, Sorina Ionica, and Gilles Dequen. Time-memory analysis of parallel collision search algorithms. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2021(2):254–274, 2021.

[35] Paul C. van Oorschot and Michael J. Wiener. Parallel collision search with application to hash functions and discrete logarithms. In Dorothy E. Denning, Raymond Pyle, Ravi Ganesan, and Ravi S. Sandhu, editors, *ACM CCS 94: 2nd Conference on Computer and Communications Security*, pages 210–218, Fairfax, Virginia, USA, November 2–4, 1994. ACM Press.

[36] Paul C. van Oorschot and Michael J. Wiener. Parallel collision search with cryptanalytic applications. *Journal of Cryptology*, 12(1):1–28, January 1999.

[37] Jacques Vélu. Isogénies entre courbes elliptiques. In *Comptes Rendus Hebdomadaires des Séances de l'Académie des Sciences*, volume Série A, page 273:238–241, 1971.
[38] Benjamin Wesolowski. The supersingular isogeny problem in time and memory $p^{1/3+o(1)}$. Cryptology ePrint Archive, Paper 2026/1486, 2026.

## A Functional graph result

In this section we show that when working in a functional graph we can add exactly one arc per weakly connected component to get a graph where each connected component has exactly two independent cycles.

**Lemma 6.** *Let $G = (V, A)$ be a functional graph. Let us denote by $G_1, \ldots, G_M$ its $M$ connected components, by $A_1, \ldots, A_M$ the arc multiset of the corresponding component, and by $v_i$ an arbitrary vertex of a component $G_i$ for all $i \in \{1, \ldots, M\}$. Suppose that $G' = (V, A')$ is a graph that contains $G$ with the property that $A' = A \cup (v_1, w_1) \cup \ldots \cup (v_M, w_M)$ for arbitrary $w_i \in V$. Then all connected components of $G'$ contain at least two independent undirected cycles.*

*Proof.* Consider a connected component of $G'$, it must contain at least one $G_i$ for some $i \in \{1, \ldots, M\}$ with an extra arc $(v_i, w_i)$ (compared to $G$). We want to prove that it has at least two independent undirected cycles. One non-trivial cycle in $G_i$ is already guaranteed by Lemma 3. We will now distinguish all cases where $(v_i, w_i)$ can land:

1. $w_i \notin G_i$: Then $(v_i, w_i)$ is an arc connecting two distinct connected components of $G$, each containing at least one non-trivial cycle. Hence, this connected component of $G'$ contains two of the $G_i$ and has at least two independent cycles (independent because they are non-trivial and disjoint).
2. $w_i \in G_i$: We again distinguish two cases:
   (a) $(v_i, w_i) \notin A_i$: If $(v_i, w_i)$ is a new arc that is added to the component, then there already exists a (possibly undirected) path from $w_i$ to $v_i$ not containing $(v_i, w_i)$ itself. So combining this path with $(v_i, w_i)$ will give us a new cycle, independent from the one that already existed in this component (since $(v_i, w_i)$ did not exist in the original component).
   (b) $(v_i, w_i) \in A_i$: In this case, we have two arcs now from $v_i$ to $w_i$, but they represent different relations. We will denote the original as $(v_i, w_i)'$. We again distinguish two cases, let $C_i$ contain all arcs that are part of the existing cycle in $G_i$:
      i. $(v_i, w_i) \notin C_i$: In this case, the arcs $(v_i, w_i)$ and $(v_i, w_i)'$ in the undirected case form a cycle, and we still have the cycle consisting of $C_i$. So in this case we also have at least two independent cycles.
      ii. $(v_i, w_i) \in C_i$: Then we can create two cycles. One is the original, containing the original arc between $v_i$ and $w_i$. The other one is the same cycle apart for taking the newly found arc between $v_i$ and $w_i$ which again gives us two independent cycles.

□

The worst-case execution is when all arc additions fall into case 2, that is, when none connect two components. Hence, in the worst case, we need to add $M$ extra arcs. This still means that we only need to add the same number of arcs as the number of connected components in a functional graph, which is $1/2\log(L)$ for our multi-user functional graph of $L$ vertices (see Lemma 2).

## B Using endomorphisms on $\mathbb{F}_{p^2}$ case

In this section we explain in detail how to use endomorphims in the $\mathbb{F}_p^2$ case. Algorithm 1 shows how we do this. This algorithm contains the following subroutines:

- $\mathsf{getArc}(E_i)$, which denotes the search for a relation on $E_i$ using the collision-style attack. It returns $E_j$ and a representation of an isogeny from $E_i$ to $E_j$, which then can be added as an arc in the multi-user graph.
- $\mathsf{findCycles}(G)$, which takes a connected component $G$ of a graph and returns the two cycles found in this component.
- $\mathsf{extractEnd}(c)$, which given a cycle $c$ uses the found isogenies between the elements of the cycle to create an endomorphism. We denote the output $\alpha_{m,E_n}$ as the $m$th endomorphism found defined on curve $E_n$.
- $\mathsf{pushEnd}(\alpha, E_i)$, which given an endomorphism $\alpha$ of a specific elliptic curve $E$ in the component pushes this endomorphism through to $E_i$ provided that it is in the same component.
- $\mathsf{checkInd}(\alpha_1, \alpha_2)$, which given two endomorphisms on the same elliptic curve checks if they are independent.
- $\mathsf{findEndRing}(\alpha_1, \alpha_2)$, which given two independent endomorphisms on the same curve outputs the endomorphism ring as in Algorithm 2.
- $\mathsf{pushEnd}(End, E_i)$, which given an endomorphism ring $End$ of a specific elliptic curve $E$ in the component pushes this endomorphism through to $E_i$ provided that it is in the same component.

Lines 3-5 simply create the multi-user graph as a functional graph. Then in lines 8-14 we continue adding arcs to this graph until every component has at least two cycles. Crucially, these loops are executed in parallel, in any architecture (client-server or centralized) that makes sure there is always at most one thread executing $\mathrm{getArc}(E_i)$ for a given $i$. In lines 17-21 we find the two cycles in a component and we push them through to the same curve (so we can check independence). Then in line 22 we check if the found endomorphisms are actually independent, if so we can find the endomorphism ring and push this ring through to the other elements in the component. If not, we add $\perp$ to the results.

For line 22 we need to be able to check independence of endomorphisms. Independence of endomorphism is defined as follows.

**Definition 2.** *Let $\alpha$ and $\beta$ be two non-trivial endomorphisms of $E$. Then $\alpha$ and $\beta$ are independent if $\alpha\beta \neq \beta\alpha$.*

**Algorithm 1** Finding Two Independent Endomorphisms

**Input:** Supersingular elliptic curves $E_1, \ldots, E_L$
**Output:** For each $i = 1, \ldots, L$ return $\mathrm{End}(E_i)$ or $\perp$

$V \leftarrow [E_1, \ldots, E_L]$ // List of vertices
$A \leftarrow []$ // List of arcs
**for** each $E_i$ in $V$ **do**
    $(E_j, \phi_{i,j}) \leftarrow \mathsf{getArc}(E_i)$
    $A.append((E_i, E_j))$
$G \leftarrow ([E_1, \ldots, E_L], A)$ // Create the functional graph
$\mathcal{C} \leftarrow \{G_1, \ldots, G_M\}$ // List the connected components of $G$
**for** each $G_n$ in $\mathcal{C}$ **do**
    $E_i \xleftarrow{\$} G_n$ // Add one arc per component
    $(E_j, \phi_{i,j}) \leftarrow \mathsf{getArc}(E_i)$
    $A.append((E_i, E_j))$
$\mathcal{C} \leftarrow \{G_1, \ldots, G_M\}$ // List the connected components of the updated $G$
**for** each $G_n$ in $\mathcal{C}$ **do**
    $(c_1, c_2) \leftarrow \mathsf{findCycles}(G_n)$ // Extract the two cycles in one component
    **for** $m = 1, 2$ **do**
        $E_m \xleftarrow{\$} c_m$ // Pick a random elliptic curve in the cycle
        $\alpha_{m,E_m} \leftarrow \mathsf{extractEnd}(c_m, E_m)$ // Extracts the endomorphism from the cycle
    $\alpha_{2,E_1} \leftarrow \mathsf{pushEnd}(\alpha_{2,E_2}, E_1)$ // Push second endomorphism through to the first curve
    **if** $\mathsf{checkInd}((\alpha_{1,E_1}, \alpha_{1,E_1}))$ **then** // Checks if the endomorphisms are independent
        $End \leftarrow \mathsf{findEndRing}(\alpha_1, \alpha_2)$ // Find the endomorphism ring
        **for** each $E_i$ in $G_n$ **do**:
            res.$append(\mathsf{pushEnd}(End, E_i))$ // Push through the found End to all curves
    **else**
        **for** each $E_i$ in $G_n$ **do**:
            res.$append(\perp)$
**return** res

Equivalent definitions could state:

- $\beta \notin \mathbb{Q}(\alpha)$.
- $\alpha \notin \mathbb{Q}(\beta)$.
- $\text{rank}(\mathbb{Z}[1, \alpha, \beta, \alpha\beta]) = 4$.

Simply testing if $\alpha\beta \neq \beta\alpha$ is the cheapest option in this case.

We now explain how findEndRing$(\alpha_1, \alpha_2)$ extracts the endomorphism ring from the two independent endomorphisms. We do this through the use of the algorithms $SATURATE_\ell$ and $SATURATERAM$ which were introduced in [30]. The algorithm $SATURATE_\ell(R_0)$ turns an order into a super-order which is maximal at $\ell$; $SATURATERAM(R_0)$ turns an order into a super-order which is maximal at $p$. The complete algorithm is Algorithm 2. It runs in subexponential time because it requires the factorization of the index of the order.

**Algorithm 2** Computing $\text{End}(E)$ from 2 endomorphism

**Input:** supersingular elliptic curve $E$, two independent endomorphisms $\alpha, \beta \in \text{End}(E)$
**Output:** $\text{End}(E)$

1: $R \leftarrow \mathbb{Z}[1, \alpha, \beta, \alpha\beta] \subseteq B_{p,\infty}$
2: $N \leftarrow \sqrt{\text{disc}(R)}/p$
3: Factor $N = p^{e_0}\ell_1^{e_1} \cdots \ell_k^{e_k}$
4: $R \leftarrow \text{SATURATERAM}(R)$
5: **for** $i = 1, \ldots, k$ **do**
6: $\quad R \leftarrow \text{SATURATE}_{\ell_i}(R)$
7: **return** $R$

**Probability of independent endomorphisms.** Now we need to analyze the probability that this algorithm will not give us what we need to break all the public keys. We claim, heuristically, that for every $i = 1, \ldots, L$ Algorithm 1 returns $\perp$ with small probability, that is the two endomorphisms $\alpha_1$ and $\alpha_2$ generated are non-trivial and independent with large probability. This probability will be dependent on the degree of the endomorphism which is a concatenation of many isogenies found in the collision attacks. We assume here that these degrees will be large.

We will show this by looking at our endomorphism as they are created, i.e., as cycles in the $\ell$-isogeny graph. We can look at all possible cycles created like this and see what the probability is that two are independent when created this way. The probability to get an endomorphism of a certain degree $d$ is $1/d$, since when doing a non-backtracking walk each step we have $\ell$ options which we choose with probability $1/\ell$ each.

This gives us two consequences:

- Endomorphisms of higher degree are less likely to be found.

– For a given degree $d$, each endomorphism of degree $d$ is equally likely to be generated.

We now analyze Algorithm 1. Fix an index $i \in \{1, \dots, L\}$ and put $E := E_i$. Fix an $\ell$-isogeny $\phi_0$ from $E$ and an integer $t$. We say that an endomorphism $\alpha$ of $E$ is admissible if

1. $\deg(\alpha) = \ell^m$ and $m \geq t$. Hence, we can factor $\alpha = \phi_m \dots \phi_1$, with $\deg(\phi_i) = \ell$.
2. $\phi_1 \neq \phi_0$
3. $\alpha$ is non-backtracking
4. The codomain of each $\phi_i$ is not $E$ whenever $i \geq t$ and $i < m$.

Let $\mathcal{A}$ be the set of admissible endomorphisms of $E$. We expect the distribution of $\alpha \in \mathrm{End}(E)$ and $\beta \in \mathrm{End}(E)$ generated in Algorithm 1 to be close to that of two independent random variables, since they are generated in a pseudorandom fashion walking in the $\ell$-isogeny graph. Each is given by the following probability measure on $E$:

$$p : \mathrm{End}(E) \to [0, 1]$$
$$p(\alpha) = \begin{cases} 1/\deg(\alpha) & \text{if } \alpha \in \mathcal{A}, \\ 0, & \text{otherwise} \end{cases} \tag{7}$$

**Lemma 7.** *The function in Equation (7) is generated by a process that takes a random (with uniform distribution) non backtracking $\ell^t$-isogeny from $E$ (with first step different from $\phi_0$) and then takes random $\ell$-isogenies steps (each choice has probability $1/\ell$) until $E$ is encountered again for the first time. It defines a probability measure.*

*Proof.* We first point out that the process stops with probability 1 since the $F_{p^2}$-graph is finite. Let $\alpha$ be an endomorphism that is generated by the process. It must satisfy all the properties in B, hence $\alpha \in \mathcal{A}$. Then it factors as $\phi_m \dots \phi_1$ into $\ell$-isogenies. For every $i \in \{2, \dots, m\}$ the probability that $\phi_i$ is the next step after $\phi_{i-1}$ is $1/\ell$ because there are $\ell$ ways to extend it in a non backtracking way. This is true even for the first step: it has $\ell$ choices because it is any $\ell$-isogeny different from $\phi_0$. Writing $A_i$ for the event "$\phi_i$ is the $i$th isogeny generated by the process" we proved that $P(A_i|A_{i-1}) = 1/\ell$. Also, note that $P(A_i|A_{i-1}) = P(A_i|A_{i-1} \cap A_{i-2} \cap \dots \cap A_1)$ because the next isogeny depends only on the last curve and is independent from the previous states (it is a so-called Markov process). Then

$$P(A_m \cap \dots \cap A_1) = P(A_1) \prod_{i=2}^{m} P(A_i|A_{i-1}) = \frac{1}{\ell^m} = \frac{1}{\deg(\alpha)}.$$

□

Next we count the endomorphism of a given norm inside a quadratic order.

**Lemma 8.** *Let $\mathcal{O}_K$ be the ring of integers of an imaginary quadratic field $K$. Let $u$ be the number of units of $K$ and call $\tau(m)$ the function counting the divisors of $m$. For any integer $m > 0$ define the set*

$$S(m) = \{x \in \mathcal{O}_K | N(x) = m\}$$

*where we denote with $N$ the norm of $K/\mathbb{Q}$. Then $\#S(m) \leq u\tau(m)$.*

*Proof.* The proof relies on [8, Exercise 17, page 249]. The multiplicative character $\chi : \mathbb{Z} \to \{-1, 0, 1\}$ is defined in [8, page 237]. The number of ideals of $\mathcal{O}_K$ (also called integral divisors of $K$) of norm $m$ equals

$$\sum_{r|m} \chi(r)$$

Each element of $S(m)$ generates a principal ideal of $\mathcal{O}_K$ of norm $m$. Two generators of a principal ideal differ by a unit meaning that the number of generators of a principal ideal is $u$. Hence, the cardinality of $S(m)$ is $u$ times the number of *principal* ideals of norm $m$ which is at most $u$ times the number of ideals of norm $m$ (with equality only when $\mathcal{O}_K$ is a UFD):

$$\#S(m) \leq u \sum_{r|m} \chi(r)$$

Finally, $\#S(m) \leq u\tau(m)$ because $\chi(r) \leq 1$. □

With this technical result in hands, it is easy to prove an upper bound for the probability of $\alpha$ and $\beta$ being dependent.

**Lemma 9.** *Suppose that $\alpha \in \mathrm{End}(E)$ and $\beta \in \mathrm{End}(E)$ are randomly and independently drawn according to the distribution Equation (7) with parameter $t$. Then, the probability that $\alpha$ and $\beta$ are dependent is at most*

$$6\frac{\sqrt{\ell}}{\sqrt{\ell} - 1}\ell^{-t/2} \tag{8}$$

*Proof.* We prove that for any given $\alpha$ the probability of $\beta$ being dependent on $\alpha$ is at most (8). Let $K$ be the quadratic imaginary field $\mathbb{Q}[\alpha]$ and $\mathcal{O}_K$ its ring of integers. Observe that the number $u$ of units in $K$ is at most 6 (in a quadratic imaginary field each unit is a root of unity, hence the only possibilities for $u$ are 2,4 or 6). Let each endomorphism in $\mathcal{A}$ has degree a power of $\ell$ with exponent

at least $t$, then

$$\begin{aligned}
P(\beta \text{ dependent on } \alpha) &= P(\beta \in \mathbb{Z}[\alpha]) \leq P(\beta \in \mathcal{O}_K) \\
&= \sum_{n=t}^{\infty} P(\beta \in \mathcal{O}_K \text{ and } \deg(\beta) = \ell^n) \\
&= \sum_{n=t}^{\infty} \frac{1}{\ell^n} \#(\beta \in \mathcal{O}_K | \deg(\beta) = \ell^n) \text{ using Equation (7)} \\
&\leq \sum_{n=t}^{\infty} \frac{1}{\ell^n} u\tau(\ell^n) \text{ by Lemma 8} \\
&\leq 6 \sum_{n=t}^{\infty} \frac{n+1}{\ell^n} \text{ because } u \leq 6 \\
&\leq 6 \sum_{n=t}^{\infty} \frac{\ell^{n/2}}{\ell^n} = 6\ell^{-t/2} \sum_{n=0}^{\infty} \ell^{-n/2} = 6 \frac{\sqrt{\ell}}{\sqrt{\ell}-1} \ell^{-t/2}
\end{aligned}$$

□

With the heuristic that endomorphisms $\alpha$ and $\beta$ are independently generated and with the probability given by Equation (7), Lemma 9 immediately implies that Algorithm 1 returns $\perp$ for a given $i \in \{1, \ldots, L\}$ with probability at most $6\frac{\sqrt{\ell}}{\sqrt{\ell}-1}\ell^{-t/2}$. Then the probability that Algorithm 1 returns $\perp$ for all $i \in \{1, \ldots, L\}$ is at most $6\frac{\sqrt{\ell}}{\sqrt{\ell}-1}\ell^{-t/2}L$, where $t$ is most likely very large.