HyperSolver: Asymptotically and Concretely Accelerating the Delfs–Galbraith Attack using Isogeny Ladders

Lorenz Panny[1], Ryan Rueger[1,2], Alessandro Sferlazza[1], and Aleksei Udovenko[3]

lorenz@yx7.cc ryan@rueg.re
alessandro.sferlazza@tum.de aleksei@affine.group

[1] Technische Universität München, Germany
[2] IBM Research Europe, Zürich, Switzerland
[3] SnT, University of Luxembourg, Luxembourg

Abstract. We present a new variant of the memoryless Delfs-Galbraith algorithm for finding an isogeny path from any given supersingular elliptic curve to a curve with known endomorphism ring. Through existing polynomial-time reductions, our algorithm allows one to solve the supersingular endomorphism-ring problem which lies at the heart of isogeny-based cryptography. For arbitrary characteristics p our algorithm asymptotically outperforms the previous best Delfs-Galbraith variant by a logarithmic factor; and matches the best previous asymptotic for characteristics *with* favourable structure. In addition to the asymptotic analysis, we also calculate a concrete cost estimate for our algorithm in terms of finite-field operations, which indicates that our expected cost is lower than all previous Delfs-Galbraith variants, even concretely. By substituting bit-operation counts for the cost of arithmetic, we can deduce concrete bit-security estimates for isogeny-based cryptographic primitives, including (but not limited to) SQIsign. As part of the calculation of the expected cost, we also provide a novel analysis of the probability of encountering "distinguished" subgraphs in an expander graph when relying various modes of graph exploration, whose potentially counterintuitive behaviour had apparently remained unnoticed thus far. Finally, we provide a complete implementation of our algorithm(s), with the asymptotic bottleneck being attacked on GPUs, and the easier post-processing done on CPU using C++ as well as SageMath. Preliminary experimental results suggest that we can solve 100-bit instances of the problem within less than 100 GPU hours. For comparison, the current cryptanalytic record for ECDLP (which has a comparable classical attack complexity) stands at 114 bits, achieved using significantly more hardware and time.

Keywords: Isogenies · Concrete Cryptanalysis · GPU Acceleration

---

27th of August, 2026
Authors listed alphabetically; see ams.org/profession/leaders/CultureStatement04.pdf.

1 Introduction

The core hardness assumptions employed in *isogeny-based cryptography* are variants of *isogeny problems*, which ask to find one or more nontrivial isogenies between two given (supersingular) elliptic curves. As the isogeny-based signature scheme *SQIsign* [1] is currently considered for standardization in the context of the NIST call for post-quantum digital signatures [26], a good understanding of the concrete hardness of the isogeny problem that underpins the security of SQIsign is of crucial importance for assessing the scheme's precise security level.

"The" supersingular isogeny problem (SSIP) can be formulated as any one of several polynomial-time equivalent problems, including [IsogenyPath] the computation of an isogeny path $E \to E'$ between two given curves E, E', [EndRing] the computation of "enough" nontrivial endomorphisms of a given curve E, or [MaxOrder] the computation of the abstract ring structure of the endomorphism ring $\mathrm{End}(E)$ of a given curve E (see Section 2.2). Solving any of these problems is sufficient (and often necessary) to break many isogeny-based cryptosystems, including SQIsign.

Asymptotic hardness. The current state-of-the-art algorithms [10,9] to attack these problems are based on work by Delfs and Galbraith [15]. They aim at finding a smooth-degree isogeny path from the target curve E to a "special" supersingular elliptic curve E_0 defined over $\mathbb{F}_p$ whose endomorphism ring is known by construction, which then facilitates transporting the endomorphism knowledge from E_0 to E via the isogeny in polynomial time. One interpretation of the procedure underlying all variants of this approach is that they search for a large "distinguished subgraph" of the isogeny graph over $\mathbb{F}_{p^2}$ which can easily be recognized once encountered; the problem of finding an isogeny to E_0 is thereby reduced to that smaller subgraph.

Concretely, given a supersingular elliptic curve E defined over $\mathbb{F}_{p^2}$, the algorithm of [15] constructs an isogeny $E \to E_0$ in two separate phases:

(i) *"Find subgraph" phase:* Find an $\mathbb{F}_{p^2}$-isogeny $E \to E'$ whose codomain E' is defined over $\mathbb{F}_p$, by randomly walking in the supersingular ℓ-isogeny graph for a fixed small ℓ. This phase takes expected time $\widetilde{O}(\sqrt{p})$ assuming the Generalized Riemann hypothesis (GRH).

(ii) *Vectorization phase:* Find an $\mathbb{F}_p$-isogeny $E' \to E_0$. This is an isogeny path problem in an isogeny graph that is isomorphic to the Cayley graph of an imaginary-quadratic class group with discriminant in $O(p)$, which can be done in time $\widetilde{O}(\sqrt[4]{p})$ using meet-in-the-middle-style techniques and is thus asymptotically negligible.

Characteristic-independent concrete improvements. SuperSolver [10], introduced the technique of *neighbourhood inspection*, to improve on the Delfs–Galbraith algorithm by expanding the distinguished set to include not only the $\mathbb{F}_p$-curves, but also curves that are ℓ'-neighbours of an $\mathbb{F}_p$-curve, for some set of freely chosen ℓ'. Testing for membership in this distinguished subgraph is more costly,

but compensates this by having increased success probability: As long as the primes ℓ' are small enough, the tradeoff is favourable and reduces the overall *concrete*, complexity of the method.

Characteristic-dependent concrete improvements. SuperSearcher [9] improves on SuperSolver for the specific shape of field characteristics that is often used in recent isogeny-based protocols (namely, primes p for which $p+1$ or $p-1$ has a large smooth factor; in particular this includes *HD-friendly primes* of the form $p = f \cdot 2^e - 1$ with small f). First, this algorithm manages to restrict to just *ring operations*[4] in all subroutines. Second, there is a key difference in the mode of exploration of the supersingular isogeny graph: Instead of performing random non-backtracking walks in the graph using modular polynomials, SuperSearcher explores a *tree* rooted at a given curve making use of rational torsion to compute isogenies; as a result, exploration is especially cheap for field characteristics of the above shape, resulting in both a concrete as well as an asymptotic improvement by reducing the complexity of the search step to $O(p/h(p))$ operations in $\mathbb{F}_p$.[5]

Contributions.

- We propose a new memoryless technique to explore isogeny graphs using (m,n)-*isogeny ladders* [4] (see Figure 1), and show how this yields a variant HyperSolver of the Delfs–Galbraith [15] algorithm to solve the SSIP with an expected complexity of $\Theta(p/h(p))$ ring operations in $\mathbb{F}_p$. This constitutes a $\Theta(\log(p))$-factor improvement on Delfs–Galbraith and matches the complexity achieved by SuperSearcher [9], *without* imposing any restrictions on the base-field characteristic.
- Isogeny ladders require a *constant* number κ of ring operations per node visited; we further reduce the expected runtime of HyperSolver using the ℓ-*neighbourhood inspection* techniques of [10, § 5] and the $\mathbb{Z}[\sqrt{-dp}]$-*orientability* checks of [8, § 6.4]. However, κ is so small that these tools are only effective for $\ell = 2, 3, 4, d = 2, 3, 6$, universally for *all* primes p. Consequently, we obtain a conceptually straightforward algorithm that is easy to implement for all sizes and shapes of field characteristic p.
- We provide detailed analysis of how the non-independence of neighbouring curves affects the runtime of the searching phase of all Delfs–Galbraith variants; by doing so, we fill a gap in the literature on the concrete cost of the supersingular isogeny problem. For example, we note that the naive assumption that all the visited curves are independently random underestimates the expected runtime by a constant factor 3 when $p \equiv 3 \pmod 4$.
- We provide a state-of-the-art GPU-accelerated implementation of the subgraph search phase of our generalised Delfs–Galbraith variant, HyperSolver,

[4] Addition, subtraction or multiplication; importantly, not inversions or (square) roots.

[5] The expected complexity of the Delfs–Galbraith algorithm is $\widetilde{O}(p^{1/2})$ field operations: the $\widetilde{O}$ and the use of inversions and square roots hides extra polynomial factors in $\log(p)$.

for solving the endomorphism ring problem in arbitrary characteristic which is capable of solving 100-bit instances within ≈ 100 GPU hours (Nvidia L40S). Moreover, we provide a CPU-only version.
- To supplement this, we provide a C++ implementation of vectorization (with minor novel optimizations, for concrete gains at small-ish discriminants) and a SageMath library to transform solutions of the isogeny problem into solutions of the endomorphism ring problem. Together, this software delivers the first all-encompassing implementation of both an IsogenyPath and EndRing solver.
- We give precise estimates for the expected cost of key recovery for SQIsign [1]. Our analysis and experiments suggest a cost of $\approx 3.67 \cdot p^{1/2}\mathsf{M}_p = 2^{127.04}\mathsf{M}_p$ for the SQIsign level-1 parameters.

Remark 1.1 (Concurrent work). While preparing this article, we were made aware of concurrent work [11] on the concrete hardness of the isogeny problem.

Their work generalizes the Delfs–Galbraith algorithm in a different direction, by exploring the supersingular graph via a depth-first search on 2-isogeny trees, and carefully studies the new framework; they introduce a cost model that captures how non-independence of neighbouring nodes in a tree affects the expected runtime of the search. This complements our analysis in Section 3.5, where we apply similar ideas to the exploration of linear walks and (2,3)-isogeny ladders.

Moreover, their work features novel techniques to efficiently detect orientability of the visited nodes, and adapts the vectorization phase accordingly. We expect these techniques, orthogonal to the improvements of our paper, to be compatible with our algorithms. The possibility of combining the techniques of the two works for further improvements is an avenue for future work.

Remark 1.2 (New $p^{1/3+o(1)}$ attack). Very recently, Wesolowski [36] proposed a new powerful heuristic algorithm for the EndRing problem, with time and memory complexity $p^{1/3+o(1)}$. Asymptotically, this significantly improves upon the previous complexity $\widetilde{O}(p^{1/2})$, which had been widely believed to be optimal.

However, the concrete performance of the new method and its impact on the security analysis of isogeny-based schemes (e.g., SQIsign) is yet to be established. If one assumes a random access inside memory of size m to cost time $O(m^{1/2})$, then Wesolowski's attack (in its current form) costs time $p^{1/2+o(1)}$ and memory $p^{1/3+o(1)}$, which is presumably worse than Delfs–Galbraith-style attacks (which cost $\widetilde{O}(p^{1/2})$ time and $(\log p)^{O(1)}$ memory). This model of memory-access costs matches the area-time metric [33] and closely reflects the cost on current real-world hardware design. As such, it is much more realistic than the RAM model, which assumes memory-access cost $O(1)$ regardless of size, despite the latter commonly being considered in cryptanalysis as well.

We conclude that the traditional attack considered in this work remains relevant due to its low memory footprint and high concrete performance, and improving our understanding of it would also help to quantify the advantage of Wesolowski's method. Finally, we remark that many of our techniques (in particular, the use of ladders for exploring isogeny graphs) are independently useful tools and can potentially also be used to optimize the new attack.

Acknowledgments. Ryan Rueger was funded by SNSF Grant CryptonIs 213766. Aleksei Udovenko was funded by the Luxembourg National Research Fund's (FNR) project PQSeal (CORE/24/IS/18978392).

2 Preliminaries

2.1 Supersingular isogeny graphs

Let $p \geq 5$ be a prime. For $q \in \{p, p^2\}$, let $\mathcal{S}_q \subseteq \mathbb{F}_q$ denote the set of supersingular j-invariants in $\mathbb{F}_q$. We have $\#\mathcal{S}_{p^2} = \lfloor p/12 \rfloor + \delta$ with $\delta \in \{0, 1, 2\}$ and $\#\mathcal{S}_p = \lambda \cdot h(-4p)$, where the factor λ equals $1/2$ if $p \equiv 1 \pmod 4$, equals 1 if $p \equiv 7 \pmod 8$, and equals $2/3$ if $p \equiv 3 \pmod 8$. Assuming GRH, the class number can be bounded by $h(-4p) \in \Omega(\sqrt{p}/\log\log(p)) \cap O(\sqrt{p} \cdot \log\log(p))$; see [22].

For another prime $\ell \neq p$, we let $\mathcal{G}_\ell(\mathbb{F}_{p^2})$ be the *supersingular ℓ-isogeny graph*, whose vertex set is $\mathcal{S}_{p^2}$ and whose edges are ℓ-isogenies between the associated elliptic curves up to post-composition with automorphisms. We also let $\mathcal{G}_\ell(\mathbb{F}_p)$ denote the subgraph of $\mathcal{G}_\ell(\mathbb{F}_{p^2})$ induced on the vertex set $\mathcal{S}_p$.

Following [27,3], for an imaginary-quadratic field K, the *K-oriented isogeny graph* $\mathcal{G}_{K,\ell}$ has as vertices the pairs (E, ι) of supersingular elliptic curves $E/\mathbb{F}_{p^2}$ with a *K-orientation*, *i.e.* a ring embedding $\iota\colon K \hookrightarrow \mathrm{End}^\circ(E) = \mathrm{End}(E) \otimes \mathbb{Q}$, up to K-oriented isomorphism, and as edges the K-oriented ℓ-isogenies, again up to K-oriented post-automorphisms.[6] We say that an order $\mathcal{O}$ in K is *at ℓ-depth $v \geq 0$* if its conductor is of the form $\ell^v \cdot m$ with $\ell \nmid m$, and *ℓ-maximal* if $v = 0$. The same terminology applies to K-oriented curves (E, ι) by considering the order $\mathcal{O} = \iota^{-1}(\mathrm{End}(E))$. For a K-oriented ℓ-isogeny $\varphi\colon (E, \iota) \to (E', \iota')$ write $\mathcal{O} = \iota^{-1}(\mathrm{End}(E))$ and $\mathcal{O}' = \iota'^{-1}(\mathrm{End}(E'))$; then φ is called *horizontal* if $\mathcal{O}' = \mathcal{O}$, *ascending* if $\mathcal{O}' \supsetneq \mathcal{O}$, and *descending* if $\mathcal{O}' \subsetneq \mathcal{O}$. The oriented isogeny graph $\mathcal{G}_{K,\ell}$ is an infinite graph whose connected component are *volcanoes*:[7] The induced subgraph of the depth-0 nodes is called *surface*; it is either an isolated node, a pair of nodes connected by one horizontal isogeny, or a cycle of length ≥ 3 connected by horizontal ℓ-isogenies. Every surface node having δ horizontal isogenies has $\ell + 1 - \delta$ descending isogenies; a complete ℓ-ary tree of descending ℓ-isogenies is attached to each of these $\ell + 1 - \delta$ children. We write $\mathcal{G}_{\mathcal{O},\ell}$ for the subgraph of $\mathcal{G}_{K,\ell}$ induced by *all* $\mathcal{O}$-oriented curves, that is, nodes at depths 0 to v. The depth-v part of $\mathcal{G}_{\mathcal{O},\ell}$ is referred to as its *floor*.

The graph $\mathcal{G}_{K,\ell}$ *folds* onto the supersingular isogeny graph $\mathcal{G}_\ell$: Every node (E, ι) in $\mathcal{G}_{K,\ell}$ gets mapped to the node $j(E)$ in $\mathcal{G}_\ell$, and every edge $(E, \iota) \to (E', \iota')$ in $\mathcal{G}_{K,\ell}$ implies the existence of an edge $j(E) \to j(E')$ in $\mathcal{G}_\ell$. While this mapping is far from injective (it reduces an infinite graph to a finite one!), it preserves

[6] An isogeny $\varphi\colon (E, \iota) \to (E', \iota')$ between K-oriented curves is called *K-oriented* if $\varphi \circ \iota(\alpha) = \iota'(\alpha) \circ \varphi$ holds in $\mathrm{Hom}(E, E') \otimes \mathbb{Q}$ for all $\alpha \in K$.

[7] Strictly speaking, the shape of the graph around the nodes $j = 1728$ and $j = 0$ differs from this description due to their extra automorphisms. We will generally ignore this detail since it is a purely local phenomenon which thus does not impact any of our results, and discussing this issue at length would distract from the bigger picture.

adjacency, and thus the image of $\mathcal{G}_{\mathcal{O},\ell}$ inside $\mathcal{G}_\ell$ has a special shape stemming from the volcano structure of $\mathcal{G}_{\mathcal{O},\ell}$: Multiple nodes on the surface may collapse into one, but (up to a fairly large depth: almost everywhere) the trees growing from each surface node remain tree-shaped. The "depth" concept also carries over from $\mathcal{G}_{K,\ell}$ to $\mathcal{G}_\ell$: The ℓ-depth (with respect to K) of a node $j \in \mathcal{G}_\ell$ is the minimal $v \geq 0$ such that there exists some $(E, \iota) \in \mathcal{G}_{K,\ell}$ at depth v with $j(E) = j$.

In the Frobenius-oriented case, *i.e.* $K = \mathbb{Q}(\sqrt{-p})$, the nodes defined over $\mathbb{F}_p$ are those that admit a (not necessarily primitive) orientation by $\mathbb{Z}[\sqrt{-p}]$. In the case $\ell = 2$, for $p \equiv 3 \pmod 4$ the maximal order is $\mathbb{Z}[(1+\sqrt{-p})/2]$, while $\mathbb{Z}[\sqrt{-p}]$ lies at 2-depth 1; for $p \equiv 1 \pmod 4$ the order $\mathbb{Z}[\sqrt{-p}]$ is already maximal.

2.2 Equivalent isogeny problems

The hardness assumption underlying many constructions in isogeny-based cryptography, including SQIsign, is one of the following equivalent [35] problems.

Problem 2.1 (IsogenyPath). Given two supersingular elliptic curves $E, E'/\mathbb{F}_{p^2}$, produce an isogeny chain $E \to E$' whose steps have degrees in polylog(p).

Problem 2.2 (EndRing). Given a supersingular elliptic curve $E/\mathbb{F}_{p^2}$, produce four endomorphisms $\alpha_1, ..., \alpha_4$ of E in efficient representation such that

$$\mathrm{End}(E) = \mathbb{Z}\alpha_1 \oplus \mathbb{Z}\alpha_2 \oplus \mathbb{Z}\alpha_3 \oplus \mathbb{Z}\alpha_4.$$

Problem 2.3 (MaxOrder). Given a supersingular elliptic curve $E/\mathbb{F}_{p^2}$, produce four elements $a_1, ..., a_4$ of the quaternion algebra $B_{p,\infty}$ such that

$$\mathrm{End}(E) \cong \mathbb{Z}a_1 \oplus \mathbb{Z}a_2 \oplus \mathbb{Z}a_3 \oplus \mathbb{Z}a_4.$$

We say that a curve E has *known endomorphism ring* if a MaxOrder solution, or (equivalently) an EndRing solution, are available for E.

Problem 2.4 (PathToOrientableCurve). Given a supersingular elliptic curve E defined over $\mathbb{F}_{p^2}$, produce an chain $E \to \cdots \to E'$ whose steps have degrees in polylog(p), such that E' has a nonscalar endomorphism of degree ℓ or ℓp with $\ell < p/4$ lying in polylog(p).

2.3 Vectorization

Vectorization is the *group action inverse problem (GAIP)* specialized to the CM action of the class group $\mathrm{cl}(\mathcal{O})$ of an imaginary-quadratic order $\mathcal{O}$ on a set of (primitively) $\mathcal{O}$-oriented elliptic curves [27,34]. The problem asks, on input two primitively $\mathcal{O}$-oriented curves E, E', to find an (invertible) ideal $\mathfrak{a} \subseteq \mathcal{O}$ such that $[\mathfrak{a}] * E = E'$. In the context of the isogeny path problem, vectorization occurs as the second phase of Delfs–Galbraith-style algorithms for the problem, which originally considered the case of a Frobenius orientation, hence $\mathcal{O} \subseteq \mathbb{Q}(\sqrt{-p})$. In this context, by construction, the conductor of $\mathcal{O}$ is smooth, hence it is always

beneficial to first "walk up" all of the available ℓ-volcanoes [32] in order to reduce to vectorization for a smaller class group, *i.e.* we can restrict our attention to the case of $\mathcal{O}$ being a maximal order. Whilst the action of every ideal $\mathfrak{a}$ may be evaluated in polynomial time with the Clapoti algorithm [29], we require a slightly stronger version of the problem to reduce EndRing to IsogenyPath followed by vectorization.

Problem 2.5 (OrientedIsogenyPath). Given primitively $\mathcal{O}$-oriented supersingular elliptic curves E, E' over $\mathbb{F}_{p^2}$, produce an isogeny chain $E \to \cdots \to E'$ whose steps are horizontal $\mathcal{O}$-oriented isogenies of degrees in polylog(p).

The state-of-the-art algorithm for vectorization is a low-memory approach due to Galbraith and Stolbunov [18], improving ideas of [17]. The idea is to compute random oriented walks from each of the two curves E, E', recording all distinguished curves on the way (not only the endpoints of the walks). Once a collision occurs that is reached both from E and E', the isogeny path can be reconstructed. The resulting path length depends on the density of distinguished points, and thus is in a trade-off with required memory.

A crucial optimization that ideals $\mathfrak{a} \subseteq \mathcal{O}$ are (pseudorandomly) selected at every step of the walk with probability inversely proportional to their cost, ensuring that the average step cost is close to the cheapest available ideal action.

To avoid undoing previously computed steps (*e.g.* computing $[\mathfrak{a}]$ and then later $[\mathfrak{a}]^{-1}$), this method necessitates the use of `ElkiesWalk` [12] instead of a basic walk using modular polynomials, which induces a noticeable overhead.

In Section 4, we propose a new optimized meet-in-the-middle approach to the problem, which eliminates this overhead at the cost of using more memory. Whilst memory asymptotically becomes a bottleneck, our approach provides comfortable performance for our parameters. We note that applying meet-in-the-middle approach for vectorization was suggested by the designers of CSIDH [7], however no optimizations were developed. We provide a summary in Table 1.

Algorithm	Ref.	Memory	Step	Path length
Full-action collision	[17]	Low	Full action	Short
Single-action collision	[18]	Low	ElkiesStep [12]	Medium
Meet-in-the-middle	[7]/Sec.4	$\widetilde{O}(\sqrt[4]{p})$	Prime action (via roots of Φ_ℓ)	Short

Table 1: Algorithms for the $\mathbb{F}_p$-isogeny path problem. Each algorithm requires on average $O(h(p)^{1/2})$ steps of different kinds.

2.4 (Generalised) Delfs–Galbraith and neighbourhood inspection

The original formulation of the Delfs–Galbraith [15, § 3] algorithm to solve the supersingular isogeny problem uses root-finding of modular polynomials to walk through the ℓ-isogeny graph until an $\mathbb{F}_p$-curve is found.

Chenu–Smith [8, § 6.4] proposed a generalisation of this, in which one searches for $\mathbb{Z}[\sqrt{-dp}]$-oriented curves, not just $\mathbb{F}_p$-curves (*i.e.* $\mathbb{Z}[\sqrt{-p}]$-oriented). Such curves E can be detected by testing whether $\Phi_d(j(E), j(E)^p) = 0$. The efficacy of this test depends on the relative cost of evaluating Φ_d and the size of $\mathbb{Z}[\sqrt{-dp}]$, and is generally only favourable for small d (if at all).

SuperSolver [10] developed the (orthogonal) technique of *neighbourhood inspection* to amortise the cost of rootfinding at every step. Their core observation was that it is possible to detect whether the polynomial $f = \Phi_\ell(X, j(E))$ had roots in $\mathbb{F}_p$ (and therefore E ℓ-neighbours over $\mathbb{F}_p$) without computing the roots; namely by determining the degree of $\gcd(\mathrm{im}(f), \mathrm{re}(f))$, an algorithm we call RootInFp.

2.5 Graphics Processing Units

GPUs are coprocessors that are optimized for highly parallel, repetitive workloads. They comprise large arrays of *vector processors* that individually are less performant than a higher-end CPU core, but ultimately attain higher throughput across the whole GPU due to the sheer number of cores (*e.g.* 2^{14} on Nvidia L40S compared to 2^7 on AMD EPYC 9754), whilst consuming comparable amounts of energy. Vector processors operate in the *single instruction, multiple data* (SIMD) paradigm in which many cores will perform the same operation (*e.g.* addition) on an array of data simultaneously. As such, they perform best when data-dependent branching is avoided, since both branches need to be executed sequentially in general (similar to "constant-time" cryptographic code).

Another limitation of GPUs is that typically, the amount of *fast* memory (*e.g.* registers, cache) available per processing core is very small compared to contemporary CPUs, hence minimizing both the amount of required memory interactions, as well as the amount of *unpredictable* memory loads in particular, is a key concern when optimizing code for GPUs.

Due to this simpler hardware architecture, algorithms for GPUs tend to be closer to their counterparts for bare-metal implementations than algorithms for CPUs; as such, our implementation effort for GPUs represents a step towards a true hardware implementation of the cryptanalytic tools employed to attack isogeny-based cryptography. For more details on the specifics of (our) CUDA-enabled GPUs, see Section 5.

2.6 Notation

We denote addition, squaring and inversion by $\mathsf{a}, \mathsf{M}, \mathsf{S}, \mathsf{I}$, indexed by p or p^2 when inside $\mathbb{F}_p$ or $\mathbb{F}_{p^2}$, *e.g.* M_{p^2} indicates multiplication in $\mathbb{F}_{p^2}$.

3 Fast exploration of isogeny graphs

In this section, we describe our variant of the (generalised) Delfs–Galbraith algorithm, HyperSolver, that employs isogeny ladders to explore the isogeny graph in

a constant number of ring operations per curve (yielding a log-factor asymptotic improvement in general characteristic), and further improves on this in terms of concrete complexity by using neighbourhood inspection and orientability detection techniques of [10,15]. The main outcome of this section will be an optimized algorithm to solve Problem 2.4.

We recall that the general two-stage strategy of the (generalised) Delfs–Galbraith algorithm: (1) the *find subgraph* stage, in which we search for isogenies between conjugate curves; and (2) the *vectorization phase* in which we solve the isogeny path problem inside the oriented subgraph.

3.1 Asymptotic improvement: Revisiting isogeny ladders

In this subsection, we give an overview on how isogeny ladders can be implemented to walk through the isogeny graph in a constant number of ring operations per visited curve.

Isogeny pushouts. The core subroutine used to construct an isogeny ladder is a *pushout*: any pair of isogenies $E \to E_m$ and $E \to E_n$ of coprime degrees m, n gives rise to a commuting square

$$\begin{array}{ccc} E & \xrightarrow{m} & E_m \\ {\scriptstyle n}\downarrow & & \downarrow{\scriptstyle n} \\ E_n & \xrightarrow[m]{} & E_{nm} \end{array}$$

In other words, given E, E_n, E_m, there exists a curve E_{nm} which is simultaneously m-isogenous to E_n and n-isogenous to E_m; its j-invariant $j(E_{nm})$ can be computed as a root of $g = \gcd(\Phi_m(X, j(E_n)), \Phi_n(X, j(E_m)))$.

Notably, by Lemma A.1, this (monic) gcd g has degree 1 with overwhelming probability when E is a uniformly random supersingular curve and $n, m \in o(p)$.[8] This will be exactly our setting, and so $j(E_{nm})$ can be directly read off the gcd $g = X - j(E_{nm})$ with overwhelming probability.

Remark 3.1. We note that if $\deg(g) \geq 2$, we have found a curve with a non-trivial endomorphism of degree n^2m^2, and so can proceed with vectorization [24]. To simplify exposition, we will assume that all such $\gcd(\Phi_2(X, j_n), \Phi_3(X, j_m))$ have exactly one root in what follows. Our implementation, however, handles these cases.

Isogeny ladders. Given a *chain* of successively m-isogenous j-invariants $\mathcal{J}, \mathcal{J}_1^2, \ldots, \mathcal{J}_1^w$ and n-isogenous j-invariants $\mathcal{J}, \mathcal{J}_2^2, \ldots, \mathcal{J}_h^h$, we can inductively push out squares along the diagonal $\mathcal{J}_*^d$ to obtain an *isogeny ladder* from [4, Sec. 4.2], as illustrated by Figure 1.

[8] Intuitively, this is because if the degree is > 1, then E_m has an non-integer endomorphism of degree n^2m^2, which is rare when n, m are small.

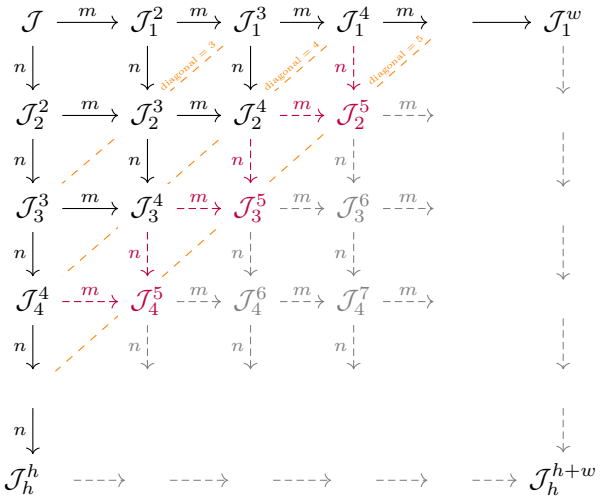


Fig. 1: An (m, n)-isogeny ladder of width w, and height h. j-invariants with an arrow labelled χ between them are χ-isogenous. The index $\mathcal{J}_t^d$ indicates the t-th curve in the d-th diagonal. The initial railings $\mathcal{J}_1^i$ (horizontal), $\mathcal{J}_i^i$ (vertical) are computed by computing successive roots of the modular polynomials Φ_m, Φ_n; the inner nodes can be computed as pushouts via gcd computations.

We immediately note that the total required memory is $\min(w, h) \cdot 2\log(p)$ bits, as only the previous diagonal must be stored to compute the next one. Indeed, the maximal length of diagonal is $\min(w, h)$, and supersingular j-invariants are defined over $\mathbb{F}_{p^2}$.

We now analyse the asymptotic cost of computing such an (m, n)-isogeny ladder of width w and height h, by analysing the cost of the initial *railings* $\mathcal{J}_1^i, \mathcal{J}_i^i$, and the cost of computing each diagonal. To save computation on the diagonals, we first compute the *inversion-free* gcds $\lambda_{t,d}(X - \mathcal{J}_t^d)$ (assumed to have degree 1, see Remark 3.1); and then recovering the exact pushouts $\mathcal{J}_t^d$ by batching the inversions $\lambda_{t,d}\,\mathcal{J}_t^d/\lambda_{t,d}$ together, so that only one inversion must be computed per diagonal.

Initial railings. The initial *railings* $\mathcal{J}_1^i, \mathcal{J}_i^i$ of the ladder can be computed by repeatedly finding random roots of the modular polynomials Φ_m, Φ_n instantiated at previously computed curves $\mathcal{J}_1^{k-1}$ and $\mathcal{J}_{k-1}^{k-1}$ and so cost an expected

$$w(O(m^2 \log(p))\mathsf{M} + \mathsf{I}) + h(O(n^2 \log(p))\mathsf{M} + \mathsf{I})$$

in total. Indeed, the complexity of $O(d^2 \log p)\mathsf{M} + \mathsf{I}$ for finding a random root of a degree-d polynomial $f \in \mathbb{F}_q[x]$ can be achieved by projectivizing the computation of $\gcd(r^{(q-1)/2} \pm 1, f)$ where r is a uniformly random degree-d polynomial; see [19, § 14.15] for details on the (non-inversion-free) method.

Instantiating modular polynomials. We recall that the ℓ-th modular polynomial Φ_ℓ in $\mathbb{F}_p[X,Y]$ has degree $\Theta(\ell)$ in both X, Y. As such, *instantiating* the modular polynomial at j (*i.e.* computing $\Phi_\ell(X, j)$) costs $O(\ell^2)\mathsf{M}$.[9]

Inversion-free gcds. SuperSolver employed an *inversion-free* gcd algorithm [10, Alg. 1] that only uses ring operations to compute the gcd *up to scalar*: requiring at most $O((\deg(f) + \deg(g))^2)$ field multiplications and additions to compute $\gcd(f, g)$ of two polynomials f, g up to an unknown scalar. When using the m-th and n-th modular polynomials, we obtain a cost of $O((n+m)^2)(\mathsf{M} + \mathsf{a})$.

Batched inversions. Finally, to obtain the pushout $\mathcal{J}_t^d$ on the diagonal $\mathcal{J}_*^d$ from each inversion-free gcd computed $\lambda_{t,d}(X - \mathcal{J}_t^d)$, we employ *Montgomery's trick* [25, 10.3.1]; allowing us to compute $\mathcal{J}_t^d = \lambda_{t,d}\mathcal{J}_t^d/\lambda_{t,d}$ for every $\mathcal{J}_t^d$ in the diagonal in $3\min(h, w)\mathsf{M} + \mathsf{I}$ in total whilst requiring $O(\min(h, w))\log(p)$ storage.

Constant ring operations per curve. The total cost for the entire $w \times h$ ladder is bounded from above by the sum of

- Railings: $w(O(m^2\log(p))\mathsf{M} + \mathsf{I}) + h(O(n^2\log(p))\mathsf{M} + \mathsf{I})$
- Instantiating modular polynomials: $wh(O(m^2) + O(n^2))\mathsf{M}$
- Inversion-free gcds: $whO((n+m)^2)\mathsf{M}$
- Batched inversions: $(w+h)(3\min(w,h)\mathsf{M} + \mathsf{I})$

As such, the expected final cost per node is then asymptotically bounded by

$$\left(\frac{O(m^2\log(p))}{h} + \frac{O(n^2\log(p))}{w}\right)\mathsf{M} + \left(\frac{2}{h} + \frac{2}{w}\right)\mathsf{I} + O((m+n)^2)\mathsf{M} \qquad (1)$$

Recalling that n, m are fixed and that the cost of one field inversion lies in $O(\log(p)) \cdot \mathsf{M}$, we finally obtain an asymptotically constant number of ring operations per visited curve by letting $w, h \in \Theta(\log(p))$. In fact, if w, h grow as $\Omega(\log(p))$, the cost per node gets arbitrarily close to just the cost of inversion-free gcds, *i.e.* $O((m+n)^2) \cdot \mathsf{M}$. We analyse this bound concretely in the next two subsections.

3.2 Concrete improvement: HyperSolver

For the lowest concrete cost, we will choose $m = 2, n = 3$. Before proceeding to the cost analysis, we first describe some (concrete) improvements; the final algorithm is described in Algorithm 2 and its control flow depicted in Figure 2.

[9] As observed in [20, Sec. 5.1], in case of parallel instantiation of Φ_ℓ at $\Omega(\ell)$ endpoints j_i, one can amortize the cost down to the quasi-optimal $\widetilde{O}(\ell)$ per endpoint j_i (note that representing $\Phi_\ell(X, j)$ requires $\ell + 1$ field elements), via polynomial multi-point evaluation. The optimization requires fast polynomial arithmetic (e.g. [5]) and is effective only for large enough ℓ; it is not useful for our (2,3)-ladder but may be useful for larger ladders in other applications.

Remark 3.2. In this subsection, we will only measure the cost of multiplications and squarings to give an overview. In Section 3.3 we will measure costs accurately including additions. For now we assume $1\mathsf{M}_{p^2} \approx 1\mathsf{S}_{p^2} \approx 3\mathsf{M}_p$.

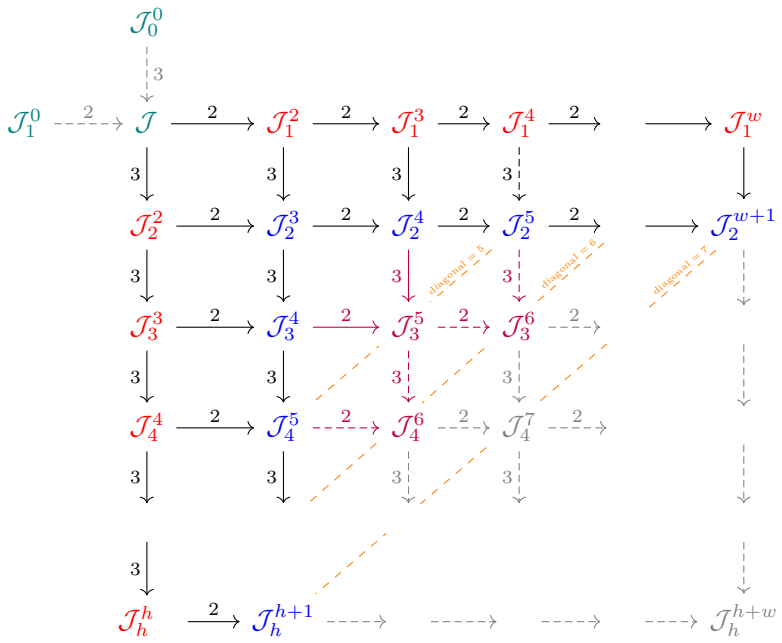


Fig. 2: The green j-invariants are the input j-invariants. The red nodes are computed by computing random roots of the successive non-backtracking modular polynomials, *i.e.* $\mathcal{J}_1^i \leftarrow \mathsf{RandomRoot}(\mathcal{N}_2(\mathcal{J}_1^{i-1}, \mathcal{J}_1^{i-2}))$. The blue nodes are computed by successively pushing out $(2,3)$-isogeny squares, using the modular-polynomials, *i.e.* $\mathcal{J}_2^i$ is the single root of $\gcd(\Phi_2(X, \mathcal{J}_2^{i-1}), \Phi_3(X, \mathcal{J}_1^{i-1}))$. The grey/purple nodes are computed successively as diagonals, using non-backtracking modular polynomials, *i.e.* $\mathcal{J}_t^d$ is the single root of $\gcd(\mathcal{N}_3(\mathcal{J}_{t-1}^{d-1}, \mathcal{J}_{t-2}^{d-2}), \mathcal{N}_2(\mathcal{J}_t^{d-1}, \mathcal{J}_t^{d-2}))$. More algorithmic details in Algorithm 2. This is the same as Figure 1, but with the inner horizontal ($\mathcal{J}_2^i$) and vertical railings ($\mathcal{J}_{i-1}^i$) computed as pushouts first, before computing along the diagonals using non-backtracking modular polynomials $\mathcal{N}_\ell$.

Using non-backtracking modular polynomials. Naïvely instantiating the modular polynomials $\Phi_\ell(X, \cdot)$ costs[10] $2(\ell^2+\ell-1)\mathsf{M}_p + \ell\mathsf{M}_{p^2}$ and computing the inversion-free gcd algorithm [10, Alg. 1] costs $((n+m+4)(n+m+5)/2-6)\mathsf{M}_{p^2}$. For $n=3, m=2$ this amounts to a total $32\mathsf{M}_p + 44\mathsf{M}_{p^2} \approx 164\mathsf{M}_p$.

[10] See `inst_ell()` in `estimator/instantiation_optimisations.py`. This is a $(3\ell+12)\mathsf{M}_p$ improvement over [10].

We are able to improve this by using *non-backtracking* modular polynomials

$$\mathcal{N}_\ell(\mathcal{J}, \mathcal{J}') := \Phi_\ell(X, \mathcal{J})/(X - \mathcal{J}')$$

whereby $\mathcal{J}, \mathcal{J}'$ are ℓ-isogenous. As the name suggests, the roots of $\mathcal{N}_\ell(j(E), j(E'))$ are all ℓ-isogenous neighbours of E that are not E'. Computing $\mathcal{N}_\ell$ costs[11] $2(\ell^2 - 1)\mathsf{M}_p + 2(\ell - 1)\mathsf{M}_{p^2}$; which is $\approx (\ell - 6)\mathsf{M}_p$ more than $\Phi_\ell(X, \cdot)$, and so is slightly cheaper for small ℓ. The real saving is due to the fact that $\mathcal{N}_\ell$ has degree one less than $\Phi(X, \cdot)$, making the inversion-free gcd cheaper. Indeed, for $m = 2, n = 3$, the inversion-free gcd costs $((n + m + 2)(n + m + 3)/2 - 6) = 22\mathsf{M}_{p^2}$, bringing the total cost of instantiation and inversion-free gcd down to $22\mathsf{M}_p + 28\mathsf{M}_{p^2} \approx 106\mathsf{M}_p$.

Degree-specific optimisations. We are able to significantly reduce the pushout cost by optimising the generic algorithms by hand for $m = 2, n = 3$. A first modest improvement is due to a reduction in the cost of computing $\mathcal{N}_2$ down to $2\mathsf{M}_p + 3\mathsf{M}_{p^2} \approx 11\mathsf{M}_p$ (from $6\mathsf{M}_p + 2\mathsf{M}_{p^2} \approx 12\mathsf{M}_p$) in Algorithm 4; and $\mathcal{N}_3$ down to $6\mathsf{M}_p + 6\mathsf{M}_{p^2} \approx 24\mathsf{M}_p$ (from $16\mathsf{M}_p + 4\mathsf{M}_{p^2} \approx 28\mathsf{M}_p$) in Algorithm 5. We will see later that $1\mathsf{M}_{p^2}$ can be shared between these computations.

The significant saving comes about from computing the inversion-free gcd symbolically (assuming the degree is 1; see Remark 3.1) from the observation

$$\lambda \gcd(x^3 + Ax^2 + Bx + C, x^2 + ax + b) = \lambda(x - \mu)$$
$$\mu = b(A - a) - C \quad \lambda = B - b - a(A - a)$$

In culmination, we are able to instantiate the $\mathcal{N}_\ell$ and compute the inversion-free gcd in a total of $8\mathsf{M}_p + 11\mathsf{M}_{p^2} \approx 41\mathsf{M}_p$.

Extra checks. We now take a more detailed look at computing each individual pushout square in Figure 1; by re-using computation to perform especially cheap ℓ-neighbourhood and orientability checks, we are able to further reduce the concrete cost. Ultimately, we will obtain the picture in Figure 3, which we explain in the remainder of this subsection.

Free neighbour. Once $\mathcal{J}_t^d$ is computed, we immediately obtain the j-invariant $\mathcal{T}_t^d$ of the remaining 2-neighbour of $\mathcal{J}_t^{d-1}$ via one $\mathbb{F}_{p^2}$-subtraction. Indeed, we have already computed $\mathcal{N}_2(\mathcal{J}_t^{d-1}, \mathcal{J}_t^{d-2}) = x^2 + ax + b$, so $\mathcal{T}_t^d = -a - \mathcal{J}_t^d$.

Note that $\mathcal{T}_t^d$ is 3-isogenous to $\mathcal{T}_{t-1}^{d-1}$: because of the existence of pushouts, $\mathcal{T}_{t-1}^{d-1}$ must be 3-isogenous to a 2-neighbour of E_t^{d-1}, which must be $\mathcal{T}_t^d$ else we have found a small endomorphism (see Remark 3.1).

[11] See `inst_ell_remove_root()` in `estimator/instantiation_optimisations.py`.

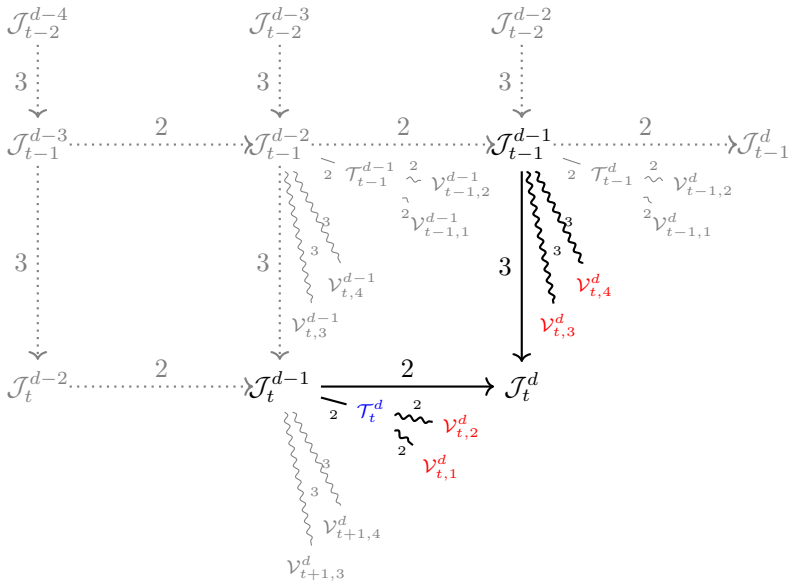


Fig. 3: The blue $\mathcal{T}_t^d$ node is not part of the ladder. The red $\mathcal{V}_{t,i}^d$ nodes are not explicitly computed, but merely inspected. The superscript indexes in which diagonal the node is explored or inspected; the subscript indexes at which height in the ladder the node is.

Neighbourhood inspection. We *inspect* the nodes $\mathcal{V}_{t,i}^d$ in Figure 3 via subfield root detection by applying RootInFp to the *inspection* polynomials

$$\mathscr{I}_{t,2}^d = \mathcal{N}_2(\mathcal{T}_t^d, \mathcal{J}_t^{d-1}) \quad \text{and} \quad \mathscr{I}_{t,3}^d = \mathcal{N}_3(\mathcal{J}_{t-1}^{d-1}, \mathcal{J}_{t-2}^{d-2})/(X - \mathcal{J}_t^d),$$

whereby $\mathscr{I}_{t,3}^d$ is computed with $\mathsf{a}_{p^2} + 1\mathsf{M}_{p^2}$

$$\mathcal{N}_3(\mathcal{J}_{t-1}^{d-1}, \mathcal{J}_{t-2}^{d-2})/(X - \mathcal{J}_t^d) = x^2 + x(A + \mathcal{J}_t^d) + B + \mathcal{J}_t^d(A + \mathcal{J}_t^d).$$

This does not reveal the $\mathcal{V}_{t,i}^d$ explicitly, but merely tests whether they are defined over $\mathbb{F}_p$; this is why we talk of *inspection* rather than exploration.

Note that we only apply RootInFp to monic quadratic polynomials, and so only costs $2\mathsf{a}_p + 1\mathsf{S}_p + 3\mathsf{M}_p$, indeed, $\mathrm{re}(f) = x^2 + ax + c$ and $\mathrm{im}(f) = dx + e$ share a root if and only if $e(e - ad) + d^2c = 0$.

Orientation detection. Similarly to neighbourhood inspection, we can also test whether $\mathcal{J}$ is $\mathbb{Z}[\sqrt{-\ell p}]$-oriented for each of the following pairs $(\mathcal{J}, \ell)$

- $(\mathcal{J}_t^{d-1}, 2)$: test whether $\mathcal{J}_t^{d-1}$ is conjugate to any of its 2-neighbours
- $(\mathcal{T}_t^d, 2)$: test whether $\mathscr{I}_{t,2}^d$ vanishes at $(\mathcal{T}_t^d)^p$

- $(\mathcal{J}_{t-1}^{d-1}, 3)$: test whether $(\mathcal{J}_{t-1}^{d-1})^p$ is equal to $\mathcal{J}_t^d$ or $\mathcal{J}_{t-2}^{d-2}$; or a root of $\mathscr{I}_{t,3}^d$.
- $(\mathcal{J}_{t-1}^{d-2}, 6)$: test whether $\mathcal{J}_{t-1}^{d-2}$ is conjugate to any of its 12 6-neighbours: (2-neighbours of $\mathcal{J}_t^{d-1}$) $\mathcal{J}_t^d, \mathcal{T}_t^d, \mathcal{J}_t^{d-2}$, (2-neighbours of $\mathcal{J}_{t-2}^{d-3}$) $\mathcal{T}_{t-2}^{d-2}, \mathcal{J}_{t-2}^{d-2}, \mathcal{J}_{t-2}^{d-4}$, (3-neighbours of $\mathcal{J}_{t-1}^{d-1}$) $\mathcal{V}_{t,3}^d, \mathcal{V}_{t,4}^d$, (3-neighbours of $\mathcal{J}_{t-1}^{d-3}$) $\mathcal{V}_{t,3}^{d-2}, \mathcal{V}_{t,4}^{d-2}$.
  We are missing the two remaining 3-neighbours of $\mathcal{T}_{t-1}^{d-1}$ (we have already checked $\mathcal{T}_t^d, \mathcal{T}_{t-2}^{d-2}$). However, we know that $\mathcal{T}_{t-1}^{d-1}$ is a 3-neighbour of $(\mathcal{J}_{t-1}^{d-2})^p$ if and only if $(\mathcal{T}_{t-1}^{d-1})^p$ is 3-neighbour of $\mathcal{J}_{t-1}^{d-2}$. This can be tested, because we know all 3-neighbours of $\mathcal{J}_{t-1}^{d-2}$ (either explicitly $\mathcal{J}_{t-2}^{d-3}, \mathcal{J}_t^{d-1}$, or implicitly $\mathcal{V}_{t,3}^{d-1}, \mathcal{V}_{t,4}^{d-1}$).
  Note that the conjugacy test of $\mathcal{J}_{t-1}^{d-2}$ with $\mathcal{J}_{t-2}^{d-4}, \mathcal{J}_{t-2}^{d-2}$ already happened in previous iterations; and the conjugacy test of $\mathcal{T}_{t-1}^{d-1}$ with $\mathcal{J}_{t-2}^{d-3}$ in a previous diagonal.
  These tests can be done for the $\mathcal{T}_\lambda^\delta$ curves by explicit comparison; and for the $\mathcal{V}_{\lambda,i}^\delta$ curves by testing whether $(\mathcal{J}_{t-1}^{d-2})^p$ is a root of $\mathscr{I}_{t-1,2}^{d-2}, \mathscr{I}_{t-1,3}^{d-2}$.

We immediately remark that $j \mapsto j^p$ is just Galois conjugation, and can be computed with $1\mathsf{a}_p$ by $\mathrm{re}(j) + \iota \cdot \mathrm{im}(j) \mapsto \mathrm{re}(j) - \iota \cdot \mathrm{im}(j)$, where $\mathbb{F}_{p^2} = \mathbb{F}_p(\iota)$; and that testing whether x is a root of $\mathscr{I}_{t,2}^d, \mathscr{I}_{t,3}^d$ amounts to evaluating a monic quadratic polynomial, costing $2\mathsf{a}_{p^2} + 1\mathsf{M}_{p^2}$ by writing $x^2 + \alpha x + \beta = x(x+\alpha) + \beta$.

More general detection. So far, we have tested for orientations by $\mathbb{Z}[\sqrt{-\ell p}]$ with small ℓ in an ad-hoc way, by re-using computation of the ladder. Employing more sophisticated techniques, it is possible to also detect orientations by $\mathbb{Z}[\ell_2\sqrt{-\ell_1 p}]$ (for small ℓ_i) [11].

Final algorithm. We can now bring together previous improvements to the laddering algorithm and give a final description of the entire HyperSolver algorithm (Algorithm 2) and the extra checks subroutine ExploreNode (Algorithm 1).

3.3 Concrete analysis: Cost of isogeny ladders

We will now analyse the cost per node, *i.e.* the constant term $O((m+n)^2)$ in Equation (1). In the next section, we will concretely compute the relationship between ladder size, memory usage, and concrete cost of the $1/w, 1/h$ terms in Equation (1). There we will see that optimal parameters dictate $w \approx 1.7\,h$, because the cost of computing more initial 3-isogenous curves is higher than computing 2-isogenous curves. For ease of exposition, we will assume this fact before proving it.

Montgomery's trick for $\mathbb{F}_{p^2}$. Using *Montgomery's trick*, one can invert every element in a set of size n with $3(n-1)\mathsf{M} + 1\mathsf{I}$. When batch-inverting elements in $\mathbb{F}_{p^2} = \mathbb{F}_p(\iota)$, it is cheaper to invert $\mathrm{re}(j) + \iota\mathrm{im}(j)$ as $(\mathrm{re}(j) - \iota\mathrm{im}(j))/(\mathrm{re}(j)^2 + \mathrm{im}(j)^2)$. As such, the total cost for batch-inverting a set of n $\mathbb{F}_{p^2}$-elements is $\mathsf{BI}_{p^2}(n) := 3(n-1)\mathsf{M}_p + \mathsf{I}_p + n(1\mathsf{a}_p + 2\mathsf{S}_p + 2\mathsf{M}_p) = n(1\mathsf{a}_p + 2\mathsf{S}_p + 5\mathsf{M}_p) - 3\mathsf{M}_p + 1\mathsf{I}_p$.

Algorithm 1 ExploreNode Exploration of ladder node $\mathcal{J}_t^d$

Input: $a \in \mathbb{F}_{p^2}$, the j-invariants $\mathcal{J}_t^d, \mathcal{T}_{t-1}^d$ and $\mathcal{J}_{t'}^{d'}$ for $d' < d$, the polynomials $\mathscr{I}_{t'}^{d'}$ for $d' < d$
Output: Success if $\mathcal{J}_t^d$ has an ℓ-neighbour over $\mathbb{F}_p$ for $\ell \in \{1,2,3,4\}$ or if it is oriented by $\mathbb{Z}[\sqrt{-mp}]$ for $m \in \{1,2,3,6\}$, failure otherwise.

Free neighbour
  $\mathcal{T}_t^d \leftarrow -a - \mathcal{J}_t^d$. ▷ $2\mathsf{a}_{p^2}$
  if $\mathcal{J}_t^d$ or $\mathcal{T}_t^d$ lie in $\mathbb{F}_p$ **return** Success: $\mathbb{F}_p$ curve
Inspect neighbourhoods
  $\mathscr{I}_{t,2}^d \leftarrow \mathcal{N}_2(\mathcal{T}_t^d, \mathcal{J}_t^{d-1})$ ▷ Algorithm 4, $3\mathsf{a}_p + 2\mathsf{M}_p + 3\mathsf{a}_{p^2} + 1\mathsf{S}_{p^2} + 2\mathsf{M}_{p^2}$
  $\mathscr{I}_{t,3}^d \leftarrow \mathcal{N}_3(\mathcal{J}_{t-1}^{d-1}, \mathcal{J}_{t-2}^{d-2})/(X - \mathcal{J}_t^d)$ ▷ Algorithm 5, $6\mathsf{a}_p + 6\mathsf{M}_p + 7\mathsf{a}_{p^2} + 1\mathsf{S}_{p^2} + 6\mathsf{M}_{p^2}$
  if RootInFp($\mathscr{I}_{t,2}^d$) or RootInFp($\mathscr{I}_{t,3}^d$) **return** Success: $\mathbb{F}_p$ curve ▷ $4\mathsf{a}_p + 2\mathsf{S}_p + 6\mathsf{M}_p$
Detect orientations
  Cache $(\mathcal{J}_t^{d-1})^p, (\mathcal{J}_{t-1}^{d-1})^p, (\mathcal{J}_{t-1}^{d-2})^p$ ▷ $3\mathsf{a}_{p^2}$
  if $(\mathcal{J}_t^{d-1})^p \in \{\mathcal{J}_t^{d-2}, \mathcal{J}_t^d, \mathcal{T}_t^d\}$ **return** Success: $\mathbb{Z}[\sqrt{-2p}]$-oriented
  if $(\mathcal{J}_{t-1}^{d-1})^p \in \{\mathcal{J}_t^d, \mathcal{J}_{t-2}^{d-2}\}$ **return** Success: $\mathbb{Z}[\sqrt{-3p}]$-oriented
  if $(\mathcal{J}_{t-1}^{d-2})^p \in \{\mathcal{J}_t^d, \mathcal{T}_t^d, \mathcal{J}_t^{d-2}, \mathcal{T}_{t-2}^{d-2}, \}$ **return** Success: $\mathbb{Z}[\sqrt{-6p}]$-oriented
  if $(\mathcal{T}_t^d)^p$ root of $\mathscr{I}_{t,2}^d$ **return** Success: $\mathbb{Z}[\sqrt{-2p}]$-oriented ▷ $3\mathsf{a}_{p^2} + 1\mathsf{M}_{p^2}$
  if $(\mathcal{J}_{t-1}^{d-1})^p$ root of $\mathscr{I}_{t,3}^d$ **return** Success: $\mathbb{Z}[\sqrt{-3p}]$-oriented ▷ $2\mathsf{a}_{p^2} + 1\mathsf{M}_{p^2}$
  Cache $\mathscr{I}_{t,3}^d$
  GetCache $\mathscr{I}_{t,3}^{d-1}, \mathscr{I}_{t,3}^{d-2}$
  if $(\mathcal{J}_{t-1}^{d-2})^p$ root of $\mathscr{I}_{t,3}^d$ or $\mathscr{I}_{t,3}^{d-2}$ **return** Success: $\mathbb{Z}[\sqrt{-6p}]$-oriented ▷ $4\mathsf{a}_{p^2} + 2\mathsf{M}_{p^2}$
  if $(\mathcal{T}_{t-1}^{d-1})^p$ root of $\mathscr{I}_{t,3}^{d-1}$ **return** Success: $\mathbb{Z}[\sqrt{-6p}]$-oriented ▷ $3\mathsf{a}_{p^2} + 1\mathsf{M}_{p^2}$
return Failure ▷ Total: $13\mathsf{a}_p + 2\mathsf{S}_p + 14\mathsf{M}_p + 27\mathsf{a}_{p^2} + 2\mathsf{S}_{p^2} + 13\mathsf{M}_{p^2}$

Inversion-free gcd of $\mathcal{N}_2, \mathcal{N}_3$. (Algorithm 2, Lines 11-20). We use Algorithm 4 and Algorithm 5 to compute $\mathcal{N}_2(\mathcal{J}_t^{d-1}, \mathcal{J}_t^{d-1}), \mathcal{N}_3(\mathcal{J}_{t-1}^{d-1}, \mathcal{J}_{t-2}^{d-2})$. However, note that $(\mathcal{J}_t^{d-1})^2$ is computed in $\mathcal{N}_2(\mathcal{J}_t^{d-1}, \mathcal{J}_t^{d-2})$, but in $\mathcal{N}_3(\mathcal{J}_t^{d-1}, \mathcal{J}_{t-1}^{d-2})$. Sharing this squaring, and adding $4\mathsf{a}_{p^2} + 2\mathsf{M}_{p^2}$ for Lines 16-18, we obtain a cost

$$\begin{aligned}&(h-2)(w-2)(9\mathsf{a}_p + 8\mathsf{M}_p + 13\mathsf{a}_{p^2} + 1\mathsf{S}_{p^2} + 9\mathsf{M}_{p^2})\\ \le\ &wh(9\mathsf{a}_p + 8\mathsf{M}_p + 13\mathsf{a}_{p^2} + 1\mathsf{S}_{p^2} + 9\mathsf{M}_{p^2}).\end{aligned}$$

Batched division. (Algorithm 2, Line 21 and 23) There are $w + h - 5$ diagonals in total; of which $w - h + 1$ have length $h - 2$ and for every length $1, \ldots, h - 3$ there are two diagonals of that length (*e.g.* the first and last diagonal have length 1). After the inversions have been computed, the division needs to be finalised (Line 23), costing a further multiplication. This brings the total cost of all batched divisions to

$$\begin{aligned}&(h-2)(w-2)\mathsf{M}_{p^2} + (w-h+1)\mathsf{BI}_{p^2}(h-2) + 2\textstyle\sum_{n=1}^{h-3}\mathsf{BI}_{p^2}(n)\\ \le\ &hw(1\mathsf{a}_p + 2\mathsf{S}_p + 5\mathsf{M}_p + \mathsf{M}_{p^2}) + (w+h)(-3\mathsf{M}_p + \mathsf{I}_p)\end{aligned}$$

Exploration. The cost of Algorithm 1 is analysed directly in the description. We bound it here to be at most

$$wh(13\mathsf{a}_p + 2\mathsf{S}_p + 14\mathsf{M}_p + 27\mathsf{a}_{p^2} + 2\mathsf{S}_{p^2} + 13\mathsf{M}_{p^2})$$

Algorithm 2: $\mathsf{HyperSolver}_{w,h}$

Computes $(2,3)$-isogeny ladder of size $w \times h$ $(w > h)$ with extra checks. Compare with Figure 2.

Input: $\mathcal{J} = \mathcal{J}_1^1, \mathcal{J}_{-1}^{-1}, \mathcal{J}_1^{-1}$, where $\mathcal{J}_{-1}^{-1}$ is 3-isogenous to $\mathcal{J}E$, and $\mathcal{J}_1^{-1}$ is 2-isogenous to $\mathcal{J}$

1: *Outer railings*
2: **for** $i = 2, \ldots, w$ **do** $\mathcal{J}_1^i \leftarrow \mathsf{RandomRoot}(\mathcal{N}_2(\mathcal{J}_1^{i-1}, \mathcal{J}_1^{i-2})$ **done** ▷ Horizontal
3: **for** $i = 2, \ldots, h$ **do** $\mathcal{J}_i^i \leftarrow \mathsf{RandomRoot}(\mathcal{N}_3(\mathcal{J}_{i-1}^{i-1}, \mathcal{J}_{i-2}^{i-2})$ **done** ▷ Vertical
4: *Inner railings*
5: **Try**
6: **for** $i = 3, \ldots, w$ **do** $\mathcal{J}_2^i \leftarrow x - \gcd(\Phi_3(X, \mathcal{J}_2^{i-1}), \Phi_2(X, \mathcal{J}_2^i))$ **done** ▷ Horizontal
7: **for** $i = 4, \ldots, h$ **do** $\mathcal{J}_{i-1}^i \leftarrow x - \gcd(\Phi_3(X, \mathcal{J}_{i-1}^{i-1}), \Phi_2(X, \mathcal{J}_i^{i-1}))$ **done** ▷ Vertical
Except Any $\deg(\gcd(\cdot,\cdot)) \geq 2$ **return** Success: Small endomorphism
8: *Diagonals*
9: **for** $d = 5, \ldots, h + w$ **do**
10: $\mathscr{D} \leftarrow []; \mathscr{N} \leftarrow []$ ▷ Initialise empty lists
11: **for** $t = 3, \ldots, \min(h, d-2)$ **do**
12: **Cache** $(\mathcal{J}_t^{d-1})^2$
13: $x^2 + ax + b \leftarrow \mathcal{N}_2(\mathcal{J}_t^{d-1}, \mathcal{J}_t^{d-2})$ ▷ Algorithm 4
14: **GetCache** $(\mathcal{J}_{t-2}^{d-1})^2$
15: $x^3 + Ax^2 + Bx + C \leftarrow \mathcal{N}_3(\mathcal{J}_{t-1}^{d-1}, \mathcal{J}_{t-2}^{d-2})$ ▷ Algorithm 5
16: $\tau \leftarrow A - a$
17: $\mathscr{N} \leftarrow \mathscr{N} + [b\tau - C]$ ▷ Append to "numerators"
18: $\mathscr{D} \leftarrow \mathscr{D} + [B - b - a\tau]$ ▷ Append to "denominators"
19: **if** $\mathscr{D}[-1] = 0$ **return** Success: Small endomorphism
20: **end for**
21: $\mathscr{I} \leftarrow \mathsf{BatchedInversion}(\mathscr{D})$
22: **for** $t = 3, \ldots, \min(h, d-2)$ **do**
23: $\mathcal{J}_t^d \leftarrow \mathscr{N}[t-3] \cdot \mathscr{I}[t-3]$
24: **if** $\mathsf{ExploreNode}(\mathcal{J}_t^d) \neq$ Failure **return** $\mathsf{ExploreNode}(\mathcal{J}_t^d)$
25: **end for**
26: **end for**

Final cost. We obtain an upper bound for the cost by taking the sum of all costs

$$wh(23\mathsf{a}_p + 4\mathsf{S}_p + 27\mathsf{M}_p + 40\mathsf{a}_{p^2} + 3\mathsf{S}_{p^2} + 23\mathsf{M}_{p^2}) + (w+h)(-3\mathsf{M}_p + \mathsf{I}_p), \quad (2)$$

whereby the cost $(w+h)(-3\mathsf{M}_p + \mathsf{I}_p)$ can be amortised to be arbitrarily small per visited node. Conversely, we obtain a lower bound by summing the cost of the inversion-free gcds (*i.e.* computing the internal nodes of the ladder) and the cost of the free neighbour and neighbourhood checks (*i.e.* Algorithm 1, Lines 1-7)

$$(w-2)(h-2)(22\mathsf{a}_p + 2\mathsf{S}_p + 22\mathsf{M}_p + 25\mathsf{a}_{p^2} + 3\mathsf{S}_{p^2} + 17\mathsf{M}_{p^2}). \quad (3)$$

We choose to exclude *orientation detection*, since it may not be worth it.

Using the approximations $6\mathsf{a}_p = 1\mathsf{M}_p = 1\mathsf{S}_p, \mathsf{a}_{p^2} = 2\mathsf{a}_p, \mathsf{S}_{p^2} = 2\mathsf{M}_p + 3\mathsf{a}_p, \mathsf{M}_{p^2} = 3\mathsf{M}_p + 5\mathsf{a}_p$ for the upper bound; and $\mathsf{a}_p = \mathsf{a}_{p^2} = 0, 2\mathsf{S}_p = \mathsf{M}_p, \mathsf{S}_{p^2} = 2\mathsf{M}_p, \mathsf{M}_{p^2} = 3\mathsf{M}_p$ for the lower bound, we obtain

$$80\mathsf{M}_p \lesssim \frac{1}{wh}\mathrm{Cost}(\mathsf{HyperSolver}_{w,h}) \lesssim 227\mathsf{a}_p + 106\mathsf{M}_p \approx 144\mathsf{M}_p. \quad (4)$$

Remark 3.3. In practice, we can decide whether or not to perform the neighbourhood and orientation tests of Algorithm 1 ahead of time, by approximating

the class numbers of $\mathbb{Z}[\sqrt{-2p}], \mathbb{Z}[\sqrt{-3p}]$, $\mathbb{Z}[\sqrt{-6p}]$. The upper-bound analysis above estimates the most expensive variant of these tests, and the lower-bound assumes that one doesn't perform any orientation detection.

3.4 Amortised cost: Ladder size and memory usage

We now investigate the concrete overhead costs from computing the outer railings, inner railings, and finally the batched inversions of the diagonals. Recall that these can be amortised to be arbitrarily small by increasing the dimensions of the ladder; we will now measure exactly what size the ladder must be, and what memory this requires.

Outer railings. (Algorithm 2, Lines 1-3) We implemented the randomised Cantor–Zassenhaus factorisation algorithm for equal-degree splitting polynomials, and experimentally[12] obtained the following expected cost estimates to compute a single root.

- *Quadratic root finding.* Each Cantor–Zassenhaus iteration costs $6\mathsf{M}_{p^2}+3\mathsf{S}_{p^2}+3\mathsf{a}_{p^2}+(14\mathsf{M}_{p^2}+8\mathsf{S}_{p^2}+12\mathsf{a}_{p^2})\log(p)$. The expected number of attempts is 2, giving a final cost per curve

$$12\mathsf{M}_{p^2}+6\mathsf{S}_{p^2}+6\mathsf{a}_{p^2}+\mathsf{I}_{p^2}+(28\mathsf{M}_{p^2}+16\mathsf{S}_{p^2}+24\mathsf{a}_{p^2})\log(p) =: \mathscr{W}_2+\mathsf{I}_{p^2}$$

- *Cubic root finding.* Each Cantor–Zassenhaus iteration costs $15\mathsf{M}_{p^2}+5\mathsf{S}_{p^2}+7\mathsf{a}_{p^2}+(42\mathsf{M}_{p^2}+12\mathsf{S}_{p^2}+40\mathsf{a}_{p^2})\log(p)$. The expected number of attempts is $4/3$, giving a final cost per curve

$$20\mathsf{M}_{p^2}+(20/3)\mathsf{S}_{p^2}+\mathsf{I}_{p^2}+(28/3)\mathsf{a}_{p^2}+(56\mathsf{M}_{p^2}+16\mathsf{S}_{p^2}+(160/3)\mathsf{a}_{p^2})\log(p) =: \mathscr{H}_3+\mathsf{I}_{p^2}$$

The inversions can all be batched together for a total cost of

$$\begin{aligned}&(w-1)\mathscr{W}_2+(h-1)\mathscr{H}_3+\mathsf{BI}_{p^2}(w+h-2)\\ \leq\ & w\mathscr{W}_2+h\mathscr{H}_3+\mathsf{BI}_{p^2}(w+h)\end{aligned}$$

Note that we will need $\geq 3h \geq w+h-5$ storage space for the diagonals at a later point in the algorithm, so this large batch of size $w+h-2$ does not increase memory consumption.

Inner railings. (Algorithm 2, Lines 5-7) Recall that to use non-backtracking modular polynomials for the pushout for E_t^d, we must know the previous curves $E_{t-2}^{d-2}, E_{t-1}^{d-1}, E_t^{d-1}, E_t^{d-2}$. This requires computing a second, "inner" initial railing E_2^i, E_i^{i+1} to start the computation (blue curves in Figure 2).

Instantiating the 2- and 3-modular polynomials costs a total of $40\mathsf{M}_p+18\mathsf{a}_{p^2}+{}+5\mathsf{M}_{p^2}$. Computing the inversion-free gcd of degree $3,4$ polynomials costs at

[12] See `estimator/cantor-zassenhaus.py`.

most $9\mathsf{a}_{p^2} + 39\mathsf{M}_{p^2}$. There are $w + h - 5$ of these inversion-free gcds to compute $\lambda(x - \mu)$; and we can recover the j-invariants explicitly by batching all necessary inversions λ^{-1} together and then multiplying $\mu \cdot \lambda^{-1}$. This brings the total cost to

$$\begin{aligned}&(w + h - 5)(40\mathsf{M}_p + 27\mathsf{a}_{p^2} + 45\mathsf{M}_{p^2}) + \mathsf{BI}_{p^2}(w + h - 5)\\ \leq\ &(w + h)(40\mathsf{M}_p + 27\mathsf{a}_{p^2} + 45\mathsf{M}_{p^2}) + \mathsf{BI}_{p^2}(w + h)\end{aligned}$$

Ladder size and amortisation. The total cost of the outer and inner railings, together with the $w + h$ inversions required for every diagonal in the ladder, is at most

$$\begin{aligned}&w\mathscr{W}_2 + h\mathscr{H}_3 + (w + h)(40\mathsf{M}_p + 27\mathsf{a}_{p^2} + 45\mathsf{M}_{p^2}) + 2\mathsf{BI}_{p^2}(w + h)\\ \leq\ &(\log(p) + 4)\left(w(24\mathsf{a}_{p^2} + 16\mathsf{S}_{p^2} + 28\mathsf{M}_{p^2}) + h(54\mathsf{a}_{p^2} + 16\mathsf{S}_{p^2} + 56\mathsf{M}_{p^2})\right) + (2 + w + h)\mathsf{I}_p\end{aligned}$$

For cryptographically sized p (*i.e.* $\log(p) \geq 2^8$), a short computation verifies that the error introduced by this estimation constitutes a proportion of at most 2^{-6}.

Since these terms are linear in w, h, but there are wh total curves in the ladder, we see that we can amortise this cost to be arbitrarily small per curve computed. To estimate optimal parameters, we assume that $6\mathsf{a}_p = 1\mathsf{M}_p = 1\mathsf{S}_p, 1\mathsf{I}_p = \log(p)\mathsf{M}_p$ and $\mathsf{a}_{p^2} = 2\mathsf{a}_p, \mathsf{S}_{p^2} = 2\mathsf{M}_p + 3\mathsf{a}_p, \mathsf{M}_{p^2} = 3\mathsf{M}_p + 5\mathsf{a}_p$. This gives a total cost of at most $(\log(p) + 4)(156w + 273h)\mathsf{M}_p$. To amortise this cost to $\mathscr{A}\mathsf{M}_p$ per node, we require $(\log(p) + 4)\,(160/h + 277/w) = \mathscr{A}$ whilst minimising the total cost. This is achieved when

$$w = 554(\log(p) + 4)/\mathscr{A} \quad \text{and} \quad h = 320(\log(p) + 4)/\mathscr{A},$$

that is a width-to-height ratio of ≈ 1.7.

Memory cost. We need to store (counted in $\mathbb{F}_{p^2}$ elements)

(i) Two initial horizontal and vertical railings $(2w + 2h)$;
(ii) Three diagonals (previous, current, next) $(3h)$
(iii) Intermediate values for each diagonal's batched inversions $(1h)$
(iv) The previous two diagonal's inspection polynomials $\mathscr{I}_{t,3}^{d-1}, \mathscr{I}_{t,3}^{d-2}$ $(2 \cdot 2h)$
(v) A (small) constant number of values during execution.

Ignoring constants, we therefore require a total of $4(w + 5h)(\log(p) + 4)\log(p) = 8616(\log(p) + 4)\log(p)/\mathscr{A}$ bits of storage.

Note that Algorithm 2, Line 14 only requires caching from the immediately previous iteration t, so only requires storing one $j(E_t^{d-1})^2$ (not h).

Example 3.4. Since the asymptotic (classical) complexity of Delfs–Galbraith is $\widetilde{O}(p^{1/2})$, isogeny schemes often pick a 256-bit prime field for a target hardness of 2^{128} (classical) operations (for example, the SQIsign level-1 prime is of size 251 bits). An amortised cost of $1\mathsf{M}_p$ (roughly 1 % of the total cost) per node would require 69 KiB of memory per ladder.

3.5 Runtime analysis

In order to estimate the expected cost of Algorithm 2, we compute the probability of encountering the "distinguished subgraph" in any given ladder as a function (1) the ladder's size (width and height) and (2) the ℓ for which ℓ-neighbourhood inspection or $\mathbb{Z}[\sqrt{-\ell p}]$-orientability checks are performed.

It was observed in [10, Rem. 5] that the clustering behaviour of the $\mathbb{F}_p$-subgraph inside $\mathcal{G}_\ell(\mathbb{F}_{p^2})$ affects the probability of encountering an $\mathbb{F}_p$-curve when enumerating $\mathcal{G}_\ell(\mathbb{F}_{p^2})$ by computing a large tree. We will show that this is true more generally for any oriented subgraph and how it clusters inside $\mathcal{G}_\ell(\mathbb{F}_{p^2})$; and also true for any method of enumerating $\mathcal{G}_\ell(\mathbb{F}_{p^2})$ that explores successive neighbours (*e.g.* linear walks, trees or ladders).

The core insight required to arrive at this result is that *isogenies transport orientation information*: if E is $\mathcal{O}$-oriented (for some imaginary quadratic order $\mathcal{O}$), and $E \to E'$ is an horizontal or ascending isogeny (with respect to $\mathcal{O}$), then E' is also $\mathcal{O}$-oriented. Recall that there are ≥ 1 horizontal or ascending isogenies, and so they constitute a considerable proportion of all possible $\ell + 1$ isogenies for small ℓ. Consequently, for small ℓ, the event of a curve E' being $\mathcal{O}$-oriented is not independent of that of an ℓ-neighbour E being $\mathcal{O}$-oriented.

Example 3.5. Consider the induced subgraph $\mathcal{G}_2(\mathbb{F}_p)$ of $\mathcal{G}_2(\mathbb{F}_{p^2})$ when $p \equiv 7 \pmod 4$ so that $\mathcal{G}_2(\mathbb{F}_p)$ is connected (*e.g.* [6, Fig. 1]). Precisely because $\mathcal{G}_2(\mathbb{F}_p)$ is connected, there are less incident edges from $\mathcal{G}_2(\mathbb{F}_{p^2}) \setminus \mathcal{G}_2(\mathbb{F}_p)$ to $\mathcal{G}_2(\mathbb{F}_p)$, and so a random walk in $\mathcal{G}_2(\mathbb{F}_{p^2})$ takes longer to encounter $\mathcal{G}_2(\mathbb{F}_p)$. Indeed, there are $2 \cdot h(-p)/2 = h(-p)$ incident edges to $\mathcal{G}_2(\mathbb{F}_p)$ (two for every node on the floor, *i.e.* at 2-depth 1), which is $1/3$ of the number of incident edges to a disconnected $\mathcal{G}_2(\mathbb{F}_{p^2})$-subgraph of size $\#\mathcal{G}_2(\mathbb{F}_p) = h(-p)$, which is $3h(-p)$. An alternative slogan to understand this, is that the *surface is only accessible through the floor*, and so there are less $\mathbb{F}_p$-curves which one can initially encounter when performing a random walk in $\mathcal{G}_2(\mathbb{F}_{p^2})$.

Previous works [10,9] estimate the cost based on the heuristic that we expect to either *explore or inspect* $\Theta(\#\mathcal{S}_{p^2}/\#\mathcal{S}_p)$ curves. In this work, in view of getting non-asymptotic cost estimates, we attempt to refine the heuristic and estimate the hidden constants.

Notation. We fix here some notation that will be used throughout the section. Let $\mathcal{O}$ be an imaginary quadratic order of discriminant Δ with field of fractions K, let $\ell \neq p$ be prime numbers, and let v be the ℓ-depth of $\mathcal{O}$. Let $\mathcal{E}_\ell(\mathcal{O})$ be the set of j-invariants in $\mathcal{S}_{p^2}$ that admit an ℓ-primitive orientation by $\mathcal{O}$, that is, that are primitively oriented by a superorder $\mathcal{O}' \supseteq \mathcal{O}$ at the same ℓ-depth as $\mathcal{O}$; write $r_{\ell,\mathcal{O}} = \#\mathcal{E}_\ell(\mathcal{O})/\#\mathcal{S}_{p^2}$. Similarly, for primes ℓ_1, ℓ_2, we denote by $\mathcal{E}_{\ell_1,\ell_2}(\mathcal{O})$ the set of j-invariants in $\mathcal{S}_{p^2}$ that are at the same ℓ_1-depth and the same ℓ_2-depth as $\mathcal{O}$, and by $r_{\ell_1,\ell_2,\mathcal{O}}$ the relative proportion in $\mathcal{S}_{p^2}$. Moreover, we let $f_{\ell,\mathcal{O}}$ denote the number of ℓ-isogenies in $\mathcal{G}_{K,\ell}$ descending from ℓ-depth v to ℓ-depth $v+1$. Note that $f_{\ell,\mathcal{O}} = \ell - \left(\frac{\Delta}{\ell}\right)$. Finally, we let $r_p = \#\mathcal{S}_p/\#\mathcal{S}_{p^2}$ denote the proportion of supersingular elliptic curves that are defined over $\mathbb{F}_p$. Note $r_{\ell,\mathbb{Z}[\sqrt{-p}]} = r_p$ for

$\ell \neq 2$, while $r_{2,\mathbb{Z}[\sqrt{-p}]} = r_p, 3r_p/4, r_p/2$ respectively when $p \equiv 1 \pmod 4, p \equiv 3 \pmod 8, p \equiv 7 \pmod 8$, since $\mathbb{Z}[\sqrt{-p}]$ may sit at 2-depth 1.

Linear walks. Below we show that the probability of hitting a special node, if we perform walk randomly in the ℓ-isogeny graph and visit n nodes, is not nr_p, but rather a constant fraction of it, depending on how the oriented isogeny volcano $\mathcal{G}_{K,\ell}$ for $K = \mathbb{Q}(\sqrt{-p})$ folds onto $\mathcal{G}_\ell$.

Lemma 3.6 (Linear-walk search for the $\mathbb{F}_p$-subgraph). *Let $\mathcal{O} = \mathbb{Z}[\sqrt{p}]$. The probability of encountering $\mathcal{S}_p$ in the first n steps of a random ℓ-isogeny walk in $\mathcal{S}_{p^2}$ with uniformly random starting curve is given by*

$$\mathcal{P} = 1 - (1 - r_p)\left(1 - \frac{f_{\ell,\mathcal{O}}}{\ell+1} r_{\ell,\mathcal{O}}\right)^n + O(n{r_{\ell,\mathcal{O}}}^2).$$

With $n \in \Theta(\log p)$, $\ell \in o(p)$, we can approximate $\mathcal{P} \approx r_p + n r_{\ell,\mathcal{O}} f_{\ell,\mathcal{O}}/(\ell+1)$. In case $p \equiv 7 \pmod 8$, we have $\mathcal{P} \approx r_p(1 + n/3)$.

Proof (Sketch). For simplicity we write $f_\ell = f_{\ell,\mathcal{O}}$ and $r_\mathcal{O} = r_{\ell,\mathcal{O}}$. For $m \in \mathbb{Z}$, denote by P_m the probability that a random non-backtracking walk of length m never hits the $\mathbb{F}_p$-subgraph, so that P_n is the complementary probability of $\mathcal{P}$. We show why the ratio P_{m+1}/P_m equals approximately $(1 - f_\ell/(\ell+1)r_\mathcal{O}) + O(r_\mathcal{O}^2)$; the statement then follows inductively after working out the base cases.

Let $\mathcal{G}$ be the subgraph of $\mathcal{G}_\ell(\mathbb{F}_{p^2})$ induced on the vertices $\mathcal{S}_{p^2} \setminus \mathcal{S}_p$. Fix $m \geq 1$ and consider a uniformly random non-backtracking path in $\mathcal{G}$ of length m. Because of the expander properties and the randomness of the starting node, we heuristically assume that the last edge of the path is approximately uniformly distributed among the edges of $\mathcal{G}$.

If a node of $\mathcal{G}$ is adjacent to the $\mathbb{F}_p$-subgraph in $\mathcal{G}_\ell(\mathbb{F}_{p^2})$, then it must be one of the $f_\ell \cdot r_\mathcal{O} \# \mathcal{S}_{p^2}$ nodes of $\mathcal{G}$ adjacent to *floor* nodes in $\mathcal{G}_\ell(\mathbb{F}_{p^2})$. Thus $\epsilon_0 = \ell \cdot f_\ell \cdot r_\mathcal{O} \# \mathcal{S}_{p^2}$ directed edges end in such a node and have probability $(\ell-1)/\ell$ of hitting the $\mathbb{F}_p$-subgraph in the next step of the non-backtracking walk. The remaining (directed) edges in $\mathcal{G}$ are $\epsilon_1 = (\ell+1) \cdot (\#\mathcal{S}_{p^2} - \#\mathcal{S}_p - f_\ell \cdot r_\mathcal{O} \# \mathcal{S}_{p^2})$, and extending the non-backtracking walk from these edges by one step stays in $\mathcal{G}$ with probability 1.

Write $\alpha = r_p/r_\mathcal{O}$. By the arguments above, we can estimate P_{m+1}/P_m, that is, the probability that a length-m walk that never hits $\mathbb{F}_p$ stays away from $\mathbb{F}_p$ in the next step, as:

$$\begin{aligned}\frac{P_{m+1}}{P_m} = \frac{\epsilon_0 \cdot (\ell-1)/\ell + \epsilon_1}{\epsilon_0 + \epsilon_1} &= \frac{(\ell+1)(\#\mathcal{S}_{p^2} - \#\mathcal{S}_p) - 2 f_\ell r_\mathcal{O} \# \mathcal{S}_{p^2}}{(\ell+1)(\#\mathcal{S}_{p^2} - \#\mathcal{S}_p) - f_\ell r_\mathcal{O} \# \mathcal{S}_{p^2}} \\ &= \frac{1 - (\alpha + 2f_\ell/(\ell+1))r_\mathcal{O}}{1 - (\alpha + f_\ell/(\ell+1))r_\mathcal{O}} = 1 - \frac{f_\ell}{\ell+1} r_\mathcal{O} + O(r_\mathcal{O}^2)\end{aligned}$$

Remark 3.7. The same arguments apply when we look for nodes that admit a (not necessarily primitive) orientation by a different quadratic order $\mathcal{O}$, e.g., of the form $\mathbb{Z}[\sqrt{-dp}]$. In this case, r_p must be replaced by the total number at ℓ-depth *smaller or equal* to the ℓ-depth of $\mathcal{O}$.

Remark 3.8 (SuperSolver's success probability). In SuperSolver, neighbourhood inspection adds an extra degree of complexity. Consider a random non-backtracking walk of length n in $\mathcal{G}_\ell(\mathbb{F}_{p^2})$, and let L be the set of primes, all distinct from ℓ, used for neighbourhood inspection. For each node j of the walk, we test whether j lies in $\mathbb{F}_p$ or is an ℓ'-neighbour of an $\mathbb{F}_p$-curve for $\ell' \in L$. For all primes ℓ, let $f_\ell := f_{\ell,\mathbb{Z}[\sqrt{-p}]}$. A similar proof as above shows that the success probability of such a walk is

$$1 - \left(1 - \gamma r_p\right) \cdot \left(1 - \gamma \frac{f_\ell}{\ell+1} r_{\ell,\mathcal{O}}\right)^n + O(nr^2) \quad \approx \quad n\gamma \frac{f_\ell}{\ell+1} r_{\ell,\mathcal{O}}$$

with $\gamma = \left(1 + \sum_{\ell' \in L} f_{\ell'}\right)$. The correcting factor γ accounts for the fact that, in this case, the nodes adjacent to the *distinguished subgraph* are not only the non-$\mathbb{F}_p$ nodes adjacent to floor nodes in $\mathcal{G}_\ell(\mathbb{F}_{p^2})$, but also the nodes connected to these ones via an ℓ'-descending isogeny.

Note that we really need to exclude the ℓ'-horizontal and ascending isogenies to avoid double-counting. Indeed, if ϕ is a ℓ'-horizontal or ascending isogeny from a floor node j to a node j', then j' is an $\mathbb{F}_p$-node too. Now let j_ℓ be a non-$\mathbb{F}_p$ node adjacent to j: one of its ℓ'-neighbours, being the (ℓ,ℓ')-pushout of (j, j', j_ℓ), is also an ℓ-neighbour of j', that is, is already counted as an ℓ-neighbour of a floor node.

Remark 3.9. Combining SuperSolver's estimates with the naive estimate that all *visited and inspected* curves are uniformly random supersingular curves, the success probability per node becomes $(1+\sum_\ell(\ell+1))r_p$, while our estimate predicts a success probability per node of $(1 + \sum_\ell f_\ell) \cdot (f_2/3) r_{2,\mathcal{O}}$.[13] For a prime $p \equiv 3 \pmod 4$, overestimates the success rate by a factor slightly larger than 3. For example, the SQIsign Level 1 prime $p = 5 \cdot 2^{248} - 1$ has optimal neighbourhood-inspecting set $\{3, 5, 7, 11\}$, and the overestimation factor is 3.48.

Ladder exploration. HyperSolver runs the following tests on the inner nodes of the ladder:

- whether $\mathcal{J}_t^d$ is f-isogenous to a curve over $\mathbb{F}_p$ for $f \in \{1, 2, 3, 4\}$.
- whether $\mathcal{J}_t^d$ lies in $\mathcal{E}(\sqrt{-Dp})$ for $D \in \{2, 3, 6\}$.

We want to estimate the probability that there exists (s, t) such that one of the above test succeeds. We estimate each one of the cases $D \in \{1, 2, 3, 6\}$ (where $D = 1$ denotes testing for $\mathbb{F}_p$-nodes) and add the success probabilities together, since random supersingular elliptic curves that are simultaneously oriented by any two distinct D's only occur with probability $O(p^{-1})$.

Proposition 3.10. *Let $\mathcal{O} = \mathbb{Z}[\sqrt{-p}]$. The probability that a $(2,3)$-isogeny ladder of size $w \times h$ ($w, h \in \Theta(\log(p))$) contains an $\mathbb{F}_p$-curve is*

$$P_{h,w} \approx 1 - \left(1 - h \frac{f_2}{3} \frac{f_3}{4} r_{2,3,\mathcal{O}}\right)^w \approx hw \frac{f_2}{3} \frac{f_3}{4} r_{2,3,\mathcal{O}}.$$

[13] See `estimator/walk_linear.py`.

Proof (Sketch). The proof proceeds as in Lemma 3.6, but at each step of a non-backtracking 2-walk we consider a whole non-backtracking 3-walk instead of a single curve. Write $r_{\mathcal{O}} = r_{2,3,\mathcal{O}}$, and $\mathcal{E}(\mathcal{O}) = \mathcal{E}_{2,3}(\mathcal{O})$.

Let P_n denote the probability that there are no $\mathbb{F}_p$-nodes in the first n columns of the ladder. Along the lines of Lemma 3.6, we show $P_{n+1}/P_n \approx 1 - \frac{f_2}{3}\frac{f_3}{4} h r_{\mathcal{O}}$. The ratio P_{n+1}/P_n represents the probability that, given a $h \times n$ ladder that never hits $\mathbb{F}_p$, the 2-pushout of the last column does not hit $\mathbb{F}_p$ either.

Denote by $T_3 \approx 1 - (1 - (f_3/4)r_{\mathcal{O}})^h$ the ratio of random non-backtracking 3-isogeny walks such that one of the curves in the walk is a curve in $\mathcal{E}(\mathcal{O})$. Out of the possible 2-isogeny pushouts of such a 3-walk, only f_ℓ pushouts are in $\mathcal{G}$,[14] *i.e.* away of the $\mathbb{F}_p$-subgraph: approximately $f_\ell \cdot T_3$ possibilities for the n-th column of the ladder are 2-adjacent to the $\mathbb{F}_p$-subgraph. With similar computations as in Lemma 3.6, we get $P_{n+1}/P_n \approx 1 - (f_2/3)T_3$. Approximating further, given $h, w \ll r_{\mathcal{O}}^{-1}$, we get $P_{n+1}/P_n \approx 1 - h(\frac{f_2}{3})(\frac{f_3}{4})r_{\mathcal{O}}$. From this follows the first equality in the statement. The last equality is again an approximation that uses $h, w \ll r_{\mathcal{O}}^{-1}$.

Remark 3.11. As done in the linear case, we can generalize the above proposition to testing orientations by a different an order $\mathcal{O}$ (as long as $\#\mathcal{E}_{2,3}(\mathcal{O})$ is comparable to the size of the $\mathbb{F}_p$-subgraph) and to neighbourhood detection (that amplifies the success probability by a factor of the form $1 + \sum_{\ell'} f_{\ell',\mathcal{O}}$).

Regarding ℓ-neighbourhood detection for one of the primes ℓ used in the exploration, the adjustment factor behaves differently. This is relevant in our algorithm, as we test whether a given node is m-isogenous to an $\mathbb{F}_p$-node for $m = 3, 4$, and the exploration primes are $2, 3$. Checking for 3-neighbours (resp. 4-neighbours) of an $\mathbb{F}_p$-node practically "descends" one level of the volcano in the $\mathbb{Z}[\sqrt{-p}]$-oriented 3-isogeny graph (resp. two levels in the oriented 2-isogeny graph). Since we replace the $\mathbb{F}_p$-floor by a lower level of larger size, namely 3 (resp. 4) times as large, the success probability gets multiplied by 3 (resp. 4) when we perform m-neighbourhood detection. As in the coprime cases, when performing m-neighbourhood detection for several values of m, the amplification factors can be summed together.

We estimate the success probability of HyperSolver as follows:

Corollary 3.12. *Let $h, w \in \Theta(\log p)$. The probability that a random non-backtracking $(2,3)$-isogeny ladder of size $w \times h$ computed by Algorithm 2 contains a curve that is $\mathcal{O}_d = \mathbb{Z}[\sqrt{-dp}]$-oriented for any $d \in \{1, 2, 3, 6\}$ is*

$$hwr + O((hwr)^2 + (h + w)r)$$

whereby

$$r = \left(\frac{7}{12} f_{2,\mathcal{O}_1} f_{3,\mathcal{O}_1} r_{2,3,\mathcal{O}_1} + \frac{f_{2,\mathcal{O}_d} f_{3,\mathcal{O}_d}}{6} r_{2,3,\mathcal{O}_d} + \sum_{d=3,6} \frac{f_{2,\mathcal{O}_d} f_{3,\mathcal{O}_d}}{12} r_{2,3,\mathcal{O}_d} \right). \quad (5)$$

[14] In a negligible fraction of all cases, there might actually be less than f_ℓ pushout, because of artifacts in the folding of the oriented volcano onto $\mathcal{G}_2(\mathbb{F}_{p^2})$.

Proof. The estimate comes by approximating the expression $1-(1-r)^{hw} = hwr + O((hwr)^2)$. Each summand $\gamma_i = f_{2,\mathcal{O}_i} f_{3,\mathcal{O}_i} r_{2,3,\mathcal{O}_i}/12$ in r comes from Proposition 3.10 (generalized to the orders $\mathcal{O}_d$); we can sum the γ_i together, *i.e.* treat $\mathcal{O}_i$-orientations independently, as curves that are simultaneously oriented by two distinct $\mathcal{O}_i$ of the above are a negligible fraction of the supersingular curves. The factor $3+4$ amplifying γ_1 comes from the fact that our exploration algorithm detects 3- and 4-neighbours of $\mathbb{F}_p$-curves: indeed, in Algorithm 1 we check all 3- and 4-neighbours of every ladder node we visit, and succeed if any of them is defined over $\mathbb{F}_p$. Similarly, γ_2 is amplified by 2 as we test whether any visited node is 2-isogenous to one oriented by $\mathbb{Z}[\sqrt{-2p}]$. The term $O((h+w)r)$ refers to the success probability of the starting initial walks.

Corollary 3.13. *As a consequence of §3.4 and §3.5: for any $\epsilon > 0$ there exist constants W and H such that* HyperSolver *with parameters $w = W\log(p)$ and $h = H\log(p)$ has expected cost at most*

$$r^{-1}p^{1/2} \cdot (215\mathsf{a}_p + 100\mathsf{M}_p + \epsilon)$$

with r as in Corollary 3.12.

4 Vectorization via optimized meet-in-the-middle

We propose an optimized meet-in-the-middle approach for the vectorization (Problem 2.5). While requiring $\widetilde{O}(\sqrt[4]{p})$ memory and some new heuristics, it is highly performant for our experimental range of ≤ 128-bit primes p; it is also conceptually simpler than the method of [18] as it does not require determining the right direction through computation of the action of the Frobenius endomorphism on the isogeny kernels (the `ElkiesWalk`). We start by defining an "action box", which is a set of elliptic curves reachable by bounded amounts of chosen ideal actions. The meet-in-the-middle algorithm will compute an action box for each of the two target elliptic curves and intersect them.

Definition 4.1 (Action box). *Let E be an $\mathcal{O}$-oriented elliptic curve. For given invertible prime ideals $\mathfrak{l}_1, \ldots, \mathfrak{l}_n \subseteq \mathcal{O}$, and bounds $B_1, \ldots, B_n \in \mathbb{Z}_{\geq 0}$, define the action box of E as the indexed (multi)set*

$$S(E; \mathfrak{l}_1, \ldots, \mathfrak{l}_n; B_1, \ldots, B_n) = \{[\mathfrak{l}_1^{e_1} \cdots \mathfrak{l}_n^{e_n}]E\}_{-B_i \leq e_i \leq B_i}.$$

The cardinality of an action box (as a multiset) is equal to $B = \prod_{i=1}^{n}(2B_i+1)$. In the following, we will focus on the Frobenius orientation for simplicity, *i.e.* $\mathcal{O} \subseteq \mathbb{Q}(\sqrt{-p})$, but the techniques should readily carry over to other orientations: A $\sqrt{-dp}$-oriented isogeny step $[\mathfrak{l}^{\pm 1}]$ from a j-invariant j can (almost always) be computed quickly as $\gcd(\Phi_\ell(X,j), \Phi_d(X,X^p))$, and pushouts using modular polynomials are (almost always) unique, so they can be computed the same way regardless of the orientation. In summary, it appears that only minor adaptations will be necessary to generalize the algorithm to non-Frobenius orientations, and we expect it to be implemented in the final version of our code.

Optimized action box computation. Algorithm 3 provides high-level pseudocode for computing an action box. We exhibit the two following features.

First, by computing the action box recursively in the decreasing order of cost of the ideal actions $\mathfrak{l}_i$, the cheaper actions are computed more often than the more expensive ones. Under properly chosen bounds B_i, the amortized complexity per visited node is then dominated by the computational cost of the cheapest action.

Second, the algorithm works directly with j-invariants and modular polynomials. This is possible since we do not need to distinguish the actions $[\mathfrak{l}_i]$ and $[\mathfrak{l}_i^{-1}]$. Starting with a j-invariant j_0 in $\mathbb{F}_p$, we (almost always) have two ℓ_i-isogenous j-invariants over $\mathbb{F}_p$ corresponding to the actions of $\mathfrak{l}_i$ and $\mathfrak{l}_i^{-1}$. These j-invariants can be found as roots of $\Phi_{\ell_i}(X, j_0)$ in $\mathbb{F}_p$. For each of these, we continue the isogeny chain deterministically by finding the unique root $X = j_k$ of $\Phi_{\ell_i}(X, j_{k-1})/(X - j_{k-2})$. On rare occasion, we may find more than two roots at the first step, or more than one root at the further steps; this is due to conjugate pairs of irrational isogenies between rational elliptic curves (which can be avoided by considering *simple* roots, *i.e.* roots of multiplicity one), or due to extra automorphisms in the cases $j = 0, 1728$. If the procedure fails for any reason, we simply skip the subtree. Therefore, such an implementation may return a strict subset of the target action box. This does not noticeably affect the overall approach, since the input curves can be rerandomized and the chances of hitting such special cases are negligible.

Remark 4.2. For $\ell = 2$, the action is meaningful only for $p \equiv 7 \pmod 8$, in which case the 2-isogeny volcano has two levels. For this case, we need to exclude the $\mathbb{F}_p$-rational 2-isogeny from j_k towards j'_k on the floor by an additional quadratic residuosity check: $\Phi(j'_k, X)/(X - j_k)$ must split over $\mathbb{F}_p$.

We analyze the complexity of the algorithm. Let c_i denote the cost of computing the action of $\mathfrak{l}_i$, which can be set to $O(\ell_i^2 \log p)$. On the top level, we will compute $2B_n$ actions of $\mathfrak{l}_n$ (each costing c_n), yielding $j([\mathfrak{l}_n^i]E)$ for $-B_n \le i \le B_n$, and perform $2B_n+1$ recursive calls. In each of those calls, we will compute $2B_{n-1}$ actions of $\mathfrak{l}_{n-1}$ (each costing c_{n-1}) and perform $2B_{n-1} + 1$ recursive calls, and so on. Therefore, the overall cost of actions is bounded from above by

$$\left(\prod_{k=1}^{n}(2B_k + 1)\right) \cdot \left(c_1 + \frac{1}{2B_1 + 1} \cdot \left(c_2 + \frac{1}{2B_2 + 1} \cdot (\ldots)\right)\right)$$

(counting one extra action per procedure call to simplify the expression). Note that $B = \prod_{k=1}^{n}(2B_k + 1)$ is the total number of curves visited. By choosing a large enough B_1, the cost can be made arbitrarily close to c_1 per visited curve. The only downside of choosing a large B_1 is the increase of the length of the final isogeny path, as well as a potential (unclear) effect on the heuristic assumption on the uniform distribution of an action box.

Parallelization. The algorithm can be easily parallelized as follows. Consider the index k of each curve E in the output of the algorithm. Given M processors, we split the curve indices $1, 2, \ldots, B$ into M contiguous segments $a_i, a_{i+1}, \ldots, b_i$.

Algorithm 3 Action box computation.

Input: Elliptic curve $E/\mathbb{F}_p$, primes $\ell_1, \ldots, \ell_n$, bounds $B_1, \ldots, B_n$
Output: an indexed subset of $S(E; \mathfrak{l}_1, \ldots, \mathfrak{l}_n; B_1, \ldots, B_n)$, where $\mathfrak{l}_i$ is an ideal whose action corresponds to an ℓ_i-isogeny

procedure ActionBox(j_0, i, path)
  if $i = 0$ **then**
    Output (path $\mapsto j_0$)
    return
  Call ActionBox($j_0, i-1$, path$||0$)
  $\{j_{-1}, j_1\} \leftarrow$ simple roots of $\Phi_{\ell_i}(X, j_0)$ in $\mathbb{F}_p$ ▷ $j_{-1} < j_1$ for some ordering
  Call ActionBox($j_1, i-1$, path$||1$)
  for $k \in \{2, \ldots, B_i\}$ **do**
    $\{j_k\} \leftarrow$ simple roots of $\Phi_{\ell_i}(X, j_{k-1})/(X - j_{k-2})$ in $\mathbb{F}_p$
    Call ActionBox($j_k, i-1$, path$||k$)
  end for
  Call ActionBox($j_{-1}, i-1$, path$||-1$)
  for $k \in \{-2, \ldots, -B_i\}$ **do**
    $\{j_k\} \leftarrow$ simple roots of $\Phi_{\ell_i}(X, j_{k+1})/(X - j_{k+2})$ in $\mathbb{F}_p$
    Call ActionBox($j_k, i-1$, path$||k$)
  end for
end procedure
Call ActionBox($j(E), n, ()$)

Then, inside the algorithm, we skip calls to ActionBox when the segments that they process do not intersect with the one assigned to the current processor. For each prime ideal $\mathfrak{l}_i$, a processor may need to perform at most $2B_i$ "unproductive" actions needed to reach the target segment of indices. Therefore, the total overhead is at most the cost of computing $[\mathfrak{l}_1^{B_1} \ldots \mathfrak{l}_n^{B_n}]$ by each processor.

Meet-in-the-middle. In order to solve Problem 2.5, we use the ActionBox procedure twice, once per each of $j(E_1)$ and $j(E_2)$. We split the available primes in an alternating way:[15] $(\mathfrak{l}_{2k+1}, c_{2k+1})$ for j_1 and $(\mathfrak{l}_{2k}, c_{2k})$ for j_2. The isogeny path is expected to be found in the intersection of the two action boxes (as sets):

$$S(E_1; \mathfrak{l}_1, \mathfrak{l}_3, \ldots; B_1, B_3, \ldots) \cap S(E_2; \mathfrak{l}_2, \mathfrak{l}_4, \ldots; B_2, B_4, \ldots).$$

Based on the analysis above, the cost per visited curve can be made arbitrarily close to c_1 around E_1 and to c_2 per visited curve around E_2. Let $N = h(\mathcal{O}) \in \widetilde{O}(\sqrt{p})$ denote the size of the considered class group and let

$$B_L = \prod_{k=1}^{\lceil n/2 \rceil} (2B_{2k-1} + 1), \quad B_R = \prod_{k=1}^{\lfloor n/2 \rfloor} (2B_{2k} + 1)$$

denote the sizes of the two action boxes. The bounds B_i need to be chosen such that $B_L \times B_R \approx N$ for a (heuristically) reasonable success probability.

[15] This contrasts with the baby-step-giant-step approach suggested in [7], using the partition into $(\mathfrak{l}_1, \mathfrak{l}_2, \ldots, \mathfrak{l}_\nu)$ and $(\mathfrak{l}_{\nu+1}, \ldots, \mathfrak{l}_n)$ for some ν, which would lead to significantly lower performance for the second set.

The cost of computing the action boxes is $T = c_1 B_L + c_2 B_R$ (assuming B_1 and B_2 are large enough). This quantity is optimized when $c_1 B_L \approx c_2 B_R$ yielding $T \approx 2\sqrt{c_1 c_2}\sqrt{N}$, *i.e.* the cost of $2\sqrt{c_1 c_2}$ operations per visited node.

5 (GPU) Implementation

We wrote a CUDA GPU implementation and a CPU C++ implementation of subgraph finding, a C++ implementation of vectorization, and a SageMath script for solving MaxOrder from this output, available at `codeberg.org/hypersolver`.

5.1 Our hardware

Our implementation is written in CUDA C++, with some parts (optionally) optimized using inline assembly, and should compile and run on essentially all moderately recent x86_64 machines with a moderately recent Nvidia GPU. Nevertheless, for maximal throughput, we chose some parameters specific to our hardware (*e.g.* $w \times h$ ladder size): Our test environment is a server with two 128-core AMD EPYC 9754 processors, 1 TB of RAM, and two Nvidia L40S GPUs (which did the majority of the work).

Each Nvidia L40S has 142 *streaming multiprocessors* (SM), each with 4 *partitions* that govern 32 *CUDA cores* which execute in SIMD fashion. Each partition has its own scheduler capable of holding 12 *warps* at once, in effect allowing every CUDA core to be multi-threaded with 12 threads. This multithreading allows hiding the latency of memory loads. The CUDA cores natively operate on 32-bit words; this contrasts with the 64-bit architecture of modern CPUs. Each SM manages a pool of 2^{16} registers (à 32 bits) and 2^{20} bits of shared memory, which it can distribute freely to its threads.

5.2 Alternative algorithm

Before we continue discussing some of the engineering tasks associated to implementing our algorithm, we note that we implemented a slight variation of the algorithm described in Section 3: for simplicity, we do not batch inversions at any stage, nor do we cache any intermediate results.

Recall that the cost of the railings can be amortised to an arbitrarily small proportion, and that it is feasible to reduce the cost of the railings to 1 % for cryptographic parameters with very moderate memory usage (Example 3.4).

However, additional cost is borne for nodes on the inside of the ladder. Pessimistically estimating the cost of an inversion by $\log(p)\mathsf{M}_p$, we see that this incurs an overhead of a factor at most 3 against Equation (2) for 256-bit primes. Preliminary benchmarks of our arithmetic implementation suggest that the overhead is indeed less than $\log(p)\mathsf{M}_p$. We remark that modular inversions can in fact be computed in time $\widetilde{O}(\mathsf{M}_p)$, and so the lack of batching would not incur an asymptotic penalty if implemented correctly (we use the $1\mathsf{I}_p = \log(p)\mathsf{M}_p$ estimate for a worst-case analysis).

This simplifies implementation, and also allows us to compute the ladder in parallel. Indeed, after the initial railings $\mathcal{J}_1^*, \mathcal{J}_2^*, \mathcal{J}_i^i, \mathcal{J}_i^{i+1}$, and the first h diagonals have been computed, the next $w - h + 1$ diagonals have height h. To push out a diagonal, we thus employ h threads: thread t computes $\mathcal{J}_t^d$ in diagonal d. To make the most of this parallelism, we chose a fairly large ratio of width to height to increase $w - h + 1$.

5.3 Orchestration

To make full use of the hardware at our disposal, and also to simplify the GPU code, we adopted the following division of work: (1) The CPU computes initial randomised starting points along with the "outer" and "inner" railings (*i.e.* nodes $\mathcal{J}_1^*, \mathcal{J}_2^*, \mathcal{J}_i^i, \mathcal{J}_i^{i+1}$, see Figure 2); and (2) the GPU fills in the remainder of the ladder in parallel (albeit without *storing* the entire ladder). We tuned the ladder size at different parameters so that the GPUs would be fully utilised, and the average CPU load is low.

More precisely, execution begins with c CPU cores performing a random walk in the 2-isogeny graph from the input curve E to a curve E_c and computing the outer and inner railings of the ladder anchored at E_c (Figure 2). These railings are stored into a queue in RAM, and then dispatched to the two GPUs asynchronously, by first copying into GPU memory, and then calling the GPU kernel to fill out the ladder in parallel. If the GPU finds a solution E', its location in the ladder is written to GPU memory, then copied to CPU memory, from where the CPU reconstructs the path taken $E_c \to E'$.

When E' is defined over $\mathbb{F}_p$, our current implementation supports computing the vectorization to a curve with known endomorphism ring E_0, finally returning a smooth-degree isogeny from the input curve $E \to E_c \to E' \to E_0$. This is all managed by a single orchestration script, that takes as input the curve E, and returns a list of j invariants and the (small) degrees of the connecting isogenies. From this output, we have further SageMath scripts to compute MaxOrder.

5.4 Computation-memory tradeoff

To achieve maximal *occupancy* in our GPU (that is, to have every CUDA core scheduled with 12 threads simultaneously) every CUDA core can only require a maximum of $2^{16}/128/12 = 42$ 32-bit registers. This imposes serious algorithmic constraints, for at 256 bits this means only ≈ 2 $\mathbb{F}_{p^2}$-elements can be held in (register) cache at any given time. For example, the easy optimisation of caching all powers of a j-invariant to instantiate multiple modular polynomials causes many register spills, and calls for shuffling a lot of data between various levels of cache.

5.5 Results

100-bit record computation. In an example run, our software was able to find an isogeny path from a randomly generated supersingular curve $E/\mathbb{F}_{p^2}$ in charac-

teristic $p \approx 2^{100}$ to a curve defined over $\mathbb{F}_p$ with known endomorphism ring in less than 100 GPU-hours (*i.e.* ≈ 40 hours with hardware described Section 5.1).

Throughput measurements. Experiments suggest an empirical throughput of about 2^{ϱ} explored ladder nodes per second and GPU in our implementation, where $\varrho \approx 29.8, 29.0, 28.1, 27.5, 26.8, 25.7$ for characteristics p requiring $3, 4, ..., 8$ machine words (à 32 bits) respectively. This scaling looks approximately quadratic in the bit length for the smaller sizes, matching the complexity of our current multiplication algorithms; however, at some point the scaling appears to become superquadratic, which might indicate a slowdown due to the increased memory pressure at larger sizes (cf. Section 5.4).

6 Applications

6.1 Solving MaxOrder and EndRing

In this section we show how the algorithms of the previous sections, with the help of polynomial-time post-processing, yield an algorithm to solve the MaxOrder and EndRing problems (Problems 2.2 and 2.3).

Remark 6.1. A different approach to computing the endomorphism ring of a supersingular curve E was proposed in [16]. In this approach, finding a finite-index suborder of the endomorphism ring requires at least two sufficiently independent solutions to Problem 2.4, i.e., two linearly independent paths $E \to E_1$ and $E \to E_2$ to curves E_i oriented by $\mathbb{Z}[\sqrt{-d_i p}]$ for some $d_i \geq 1$. We decide not to follow the approach of [16] since the Delfs–Galbraith approach requires only one solution to Problem 2.4 (followed by vectorization and post-processing, of negligible complexity) to recover full information about the endomorphism ring, therefore our algorithm is faster by an asymptotically constant factor.

Given a supersingular elliptic curve E defined over $\mathbb{F}_{p^2}$, HyperSolver constructs another curve E_0 with known endomorphism ring and finds a smooth isogeny $\phi\colon E_0 \to E$ as in IsogenyPath (Problem 2.1).

Solving MaxOrder. Recall that $\mathcal{Q}_0 = \mathrm{End}(E_0)$ is a maximal order inside the algebra $\mathrm{End}(E_0) \otimes \mathbb{Q} \cong B_{p,\infty}$. Given an isogeny path $\phi\colon E_0 \to E$, we can compute the *connecting* (left) $\mathcal{Q}_0$-ideal $I = \mathrm{Hom}(E, E_0)\phi \subseteq \mathcal{Q}_0$ associated to ϕ (details below); from this, a maximal order $\mathcal{Q} \cong \mathrm{End}(E) \subseteq B_{p,\infty}$ can be computed as the right order of I.

Solving EndRing. From $\mathcal{Q}$, we can extract efficient isogeny representations for a list of endomorphisms comprising a basis of $\mathrm{End}(E)$. Using the connecting N-isogeny ϕ, we get the embedding of rings

$$\iota\colon \mathrm{End}(E) \hookrightarrow \mathrm{End}(E_0) \otimes \mathbb{Q} = B_{p,\infty} \qquad \alpha \mapsto (\phi^\vee \circ \alpha \circ \phi)/N$$

from which we obtain $\mathcal{Q} = \iota(\mathrm{End}(E)) \subseteq \frac{1}{N}\mathrm{End}(E_0)$. Hence, a quaternion $\gamma \in \mathcal{Q}$ corresponds to the endomorphism $\iota^{-1}(\gamma) = (\phi \circ \gamma \circ \phi^\vee)/N$. As γ itself is of the form α/N for some $\alpha \in \mathrm{End}(E_0)$ (for which we know an efficient representation), we recover a representation for $\iota^{-1}(\gamma) \in \mathrm{End}(E)$ by dividing the composite isogeny $\phi \circ \alpha \circ \phi^\vee$ by N^2: see [30, Algorithm 3] for the case of powersmooth N and [24, Algorithm 1] for an unconditionally polynomial-time version. Applying this procedure to a $\mathbb{Z}$-basis $\{1, \gamma_1, \gamma_2, \gamma_3\}$ of $\mathcal{Q}$, yields a basis of $\mathrm{End}(E)$ in efficient representation.

Isogeny-to-ideal. The *isogeny-to-ideal* translation was described in [14, §6.2] for the prime-power-degree case. We briefly summarize (a slight generalization to arbitrary smooth degrees of) the approach.

Suppose $\phi = \phi_n \circ \cdots \circ \phi_1$ is an isogeny where every step $\phi_i \colon E_{i-1} \to E_i$ (with $E_n = E$) has prime degree $\deg \phi_i = \ell_i$. Moreover, let (P, Q) be a basis of $E_0[\ell_1]$, and $(\alpha_0, \alpha_1, \alpha_2, \alpha_3)$ a basis of $\mathrm{End}(E_0) \otimes \mathbb{Q}$. For all steps $k \in \{1, \ldots, n\}$:

- Let $\psi_{k-1} \colon E_0 \to E_{k-1}$ be an isogeny of degree coprime to ℓ_k. (If $\phi_{k-1} \circ \cdots \circ \phi_0$ happens to be coprime to ℓ_k, we can set $\psi_{k-1} = \phi_{k-1} \circ \cdots \circ \phi_0$; otherwise a suitable ψ_{k-1} can be constructed from $\phi_{k-1} \circ \cdots \circ \phi_0$ using the KLPT algorithm [21].) Denote by $\psi_k = (\psi_{k-1})^* \phi_k$ the pullback of ϕ_k to E_0 via ϕ_{k-1} [13, §4.1] and let $K_k = \ker \phi_k$.
- Translate ψ_k to an ideal J_k: for $i \in \{0, 1, 2, 3\}$ compute (x_i, y_i) such that $\alpha_i(K_k) = x_i P + y_i Q$. Using linear algebra, find a linear combination $\alpha = \sum_i a_i \alpha_i$ such that $\ker(\alpha) \cap E[\ell_i] = K_k$. Set $J_k = (\ell_k, \alpha)$.

Finally, return $I_\phi = \prod_k J_k$.

6.2 Concrete security of SQIsign

In this section, we give bounds on the time complexity of HyperSolver for the primes p used in version 2 of SQIsign [1]. Below we detail the analysis at NIST security level 1, and refer to Table 2 for the primes at higher security levels.

For all SQIsign primes, all orientation and neighbourhood tests are worth it. According to Corollary 3.12, the number of curves that need to be explored using HyperSolver (Algorithm 2) before succeeding follows a geometric distribution with parameter r given by Equation (5). For the SQIsign level-1 prime $p = 5 \cdot 2^{248} - 1$, we numerically compute $r \approx 3.835 \cdot r_p$.

Upper bound Level 1. Recall that we can amortise the costs of the railings of the ladder to $1\mathsf{M}_p$ with moderate memory resources (Example 3.4). Using Equation (4) and numerically approximating the class number to be $\#\mathrm{cl}(\mathbb{Z}[(\sqrt{-p}+1)/2]) = 2^{124.857}$ (so $r_p = 12h/p = 2^{-121.879}$), we get a final expected cost of

$$\frac{\mathrm{Cost}(\mathsf{HyperSolver}_{h,w})}{\mathbb{P}(\mathrm{Success})} \approx \frac{145\,\mathsf{M}_p}{3.835 \cdot r_p} \approx 3.89 \cdot p^{1/2}\,\mathsf{M}_p \approx 2^{127.12}\,\mathsf{M}_p. \tag{6}$$

According to NIST security considerations [26], Security Level 1 asks for a classical complexity of 2^{143} gates. Whether the SQIsign level-1 parameter set formally

respects this security criterion depends on the complexity of field arithmetic, that is, whether one can build a circuit for $\mathbb{F}_p$-multiplication in less than $2^{15.88}$ gates.

Lower bound Level 1. Disregarding amortised costs, and using the same conservative assumptions on relative cost of arithmetic made for Equation (4), we obtain a lower bound $2^{126.262}\,\mathsf{M}_p$.

Table 2: Expected cost of HyperSolver for SQIsign [1]

Level	Gates	Prime	Upper bound (M_p)
I	2^{143}	$5 \cdot 2^{248} - 1$	$3.889\sqrt{p} \approx 2^{127.12}$
III	2^{207}	$65 \cdot 2^{376} - 1$	$4.775\sqrt{p} \approx 2^{193.27}$
V	2^{272}	$27 \cdot 2^{500} - 1$	$5.655\sqrt{p} \approx 2^{254.88}$

Comparison to SuperSolver. For the same prime, our result improves on SuperSolver by a factor 6.34, excluding additions from the arithmetic cost for both algorithms. Indeed, while we use $101\mathsf{M}_p$ with a success parameter $r_p \cdot 3.835$, SuperSolver (with slightly improved algorithms for modular polynomials[16]) requires $1473\mathsf{M}_p$ per step (this includes half of a square root and inspecting 30 neighbours), and has success parameter $r = r_p \cdot 9.00$.[17]

[16] See `estimator/instantiation_optimisations.py`.
[17] See `estimator/hypersolver_vs_supersolver.py`.

References

1. Aardal, M.A., Adj, G., Aranha, D.F., Basso, A., Canales Martínez, I.A., Chávez-Saab, J., Santos, M.C., Dartois, P., De Feo, L., Duparc, M., Eriksen, J.K., Fouotsa, T.B., Filho, D.L.G., Hess, B., Kohel, D., Leroux, A., Longa, P., Maino, L., Meyer, M., Nakagawa, K., Onuki, H., Panny, L., Patranabis, S., Petit, C., Pope, G., Reijnders, K., Robert, D., Rodríguez Henríquez, F., Schaeffler, S., Wesolowski, B.: SQIsign. Tech. rep., National Institute of Standards and Technology (2024), available at https://csrc.nist.gov/Projects/pqc-dig-sig/round-2-additional-signatures 2, 4, 30, 31
2. Arpin, S., Camacho-Navarro, C., Lauter, K., Lim, J., Nelson, K., Scholl, T., Sotáková, J.: Adventures in supersingularland. Cryptology ePrint Archive, Report 2019/1056 (2019), https://eprint.iacr.org/2019/1056 35
3. Arpin, S., Chen, M., Lauter, K.E., Scheidler, R., Stange, K.E., Tran, H.T.N.: Orientations and cycles in supersingular isogeny graphs. Cryptology ePrint Archive, Report 2022/562 (2022), https://eprint.iacr.org/2022/562 5
4. Basso, A., Codogni, G., Connolly, D., De Feo, L., Fouotsa, T.B., Lido, G.M., Morrison, T., Panny, L., Patranabis, S., Wesolowski, B.: Supersingular curves you can trust. In: Hazay, C., Stam, M. (eds.) Advances in Cryptology – EUROCRYPT 2023, Part II. Lecture Notes in Computer Science, vol. 14005, pp. 405–437. Springer, Cham, Switzerland, Lyon, France (Apr 23–27, 2023). https://doi.org/10.1007/978-3-031-30617-4_14 3, 9
5. Cantor, D.G., Kaltofen, E.: On fast multiplication of polynomials over arbitrary algebras. Acta Informatica **28**(7), 693–701 (Jul 1991). https://doi.org/10.1007/BF01178683 11
6. Castryck, W., Decru, T.: CSIDH on the surface. In: Ding, J., Tillich, J.P. (eds.) Post-Quantum Cryptography - 11th International Conference, PQCrypto 2020. pp. 111–129. Springer, Cham, Switzerland, Paris, France (Apr 15–17, 2020). https://doi.org/10.1007/978-3-030-44223-1_7 20
7. Castryck, W., Lange, T., Martindale, C., Panny, L., Renes, J.: CSIDH: An efficient post-quantum commutative group action. In: Peyrin, T., Galbraith, S. (eds.) Advances in Cryptology – ASIACRYPT 2018, Part III. Lecture Notes in Computer Science, vol. 11274, pp. 395–427. Springer, Cham, Switzerland, Brisbane, Queensland, Australia (Dec 2–6, 2018). https://doi.org/10.1007/978-3-030-03332-3_15 7, 26
8. Chenu, M., Smith, B.: Higher-degree supersingular group actions. Transactions on Mathematical Cryptology **1**(2), 85–101 (Mar 2022), https://journals.flvc.org/mathcryptology/article/view/130604 3, 8
9. Chi-Domínguez, J.J.: Improved subfield curve search for specific field characteristics. Cryptology ePrint Archive, Report 2025/226 (2025), https://eprint.iacr.org/2025/226 2, 3, 20, 39, 40
10. Corte-Real Santos, M., Costello, C., Shi, J.: Accelerating the Delfs-Galbraith algorithm with fast subfield root detection. In: Dodis, Y., Shrimpton, T. (eds.) Advances in Cryptology – CRYPTO 2022, Part III. Lecture Notes in Computer Science, vol. 13509, pp. 285–314. Springer, Cham, Switzerland, Santa Barbara, CA, USA (Aug 15–18, 2022). https://doi.org/10.1007/978-3-031-15982-4_10 2, 3, 8, 9, 11, 12, 20, 40, 41
11. Corte-Real Santos, M., Herlédan Le Merdy, A., Macula, J., Meyer, M., Morrison, T., Orvis, E.: Algorithms for solving the isogeny problem with oriented elliptic curves. Preprint (2026), https://ia.cr/2026/1219 4, 15, 40, 41

12. De Feo, L., Kieffer, J., Smith, B.: Towards practical key exchange from ordinary isogeny graphs. In: Peyrin, T., Galbraith, S. (eds.) Advances in Cryptology – ASIACRYPT 2018, Part III. Lecture Notes in Computer Science, vol. 11274, pp. 365–394. Springer, Cham, Switzerland, Brisbane, Queensland, Australia (Dec 2–6, 2018). https://doi.org/10.1007/978-3-030-03332-3_14 7
13. De Feo, L., Kohel, D., Leroux, A., Petit, C., Wesolowski, B.: SQISign: Compact post-quantum signatures from quaternions and isogenies. In: Moriai, S., Wang, H. (eds.) Advances in Cryptology – ASIACRYPT 2020, Part I. Lecture Notes in Computer Science, vol. 12491, pp. 64–93. Springer, Cham, Switzerland, Daejeon, South Korea (Dec 7–11, 2020). https://doi.org/10.1007/978-3-030-64837-4_3 30
14. De Feo, L., Masson, S., Petit, C., Sanso, A.: Verifiable delay functions from supersingular isogenies and pairings. In: Galbraith, S.D., Moriai, S. (eds.) Advances in Cryptology – ASIACRYPT 2019, Part I. Lecture Notes in Computer Science, vol. 11921, pp. 248–277. Springer, Cham, Switzerland, Kobe, Japan (Dec 8–12, 2019). https://doi.org/10.1007/978-3-030-34578-5_10 30
15. Delfs, C., Galbraith, S.D.: Computing isogenies between supersingular elliptic curves over $\mathbb{F}_p$. Designs, Codes and Cryptography **78**(2), 425–440 (2016). https://doi.org/10.1007/s10623-014-0010-1 2, 3, 7, 9
16. Fuselier, J., Iezzi, A., Kozek, M., Morrison, T., Namoijam, C.: Computing supersingular endomorphism rings using inseparable endomorphisms (Apr 2025). https://doi.org/10.1016/j.jalgebra.2025.01.012, http://dx.doi.org/10.1016/j.jalgebra.2025.01.012 29, 35
17. Galbraith, S.D., Hess, F., Smart, N.P.: Extending the GHS Weil descent attack. In: Knudsen, L.R. (ed.) Advances in Cryptology – EUROCRYPT 2002. Lecture Notes in Computer Science, vol. 2332, pp. 29–44. Springer Berlin Heidelberg, Germany, Amsterdam, The Netherlands (Apr 28 – May 2, 2002). https://doi.org/10.1007/3-540-46035-7_3 7
18. Galbraith, S.D., Stolbunov, A.: Improved algorithm for the isogeny problem for ordinary elliptic curves. Appl. Algebra Eng. Commun. Comput. **24**(2), 107–131 (2013) 7, 24
19. von zur Gathen, J., Gerhard, J.: Modern Computer Algebra. Cambridge University Press, USA, 2 edn. (2003) 10
20. Ionica, S., Joux, A.: Pairing the volcano. In: ANTS. Lecture Notes in Computer Science, vol. 6197, pp. 201–218. Springer (2010). https://doi.org/10.1007/978-3-642-14518-6_18 11
21. Kohel, D., Lauter, K., Petit, C., Tignol, J.P.: On the quaternion ℓ-isogeny path problem. Cryptology ePrint Archive, Report 2014/505 (2014), https://eprint.iacr.org/2014/505 30
22. Littlewood, J.E.: On the class-number of the corpus $P(\sqrt{-k})$. Proceedings of the London Mathematical Society **s2-27**(1), 358–372 (1928). https://doi.org/10.1112/plms/s2-27.1.358 5
23. Love, J., Boneh, D.: Supersingular curves with small non-integer endomorphisms (2020), https://arxiv.org/abs/1910.03180 35
24. Merdy, A.H.L., Wesolowski, B.: The supersingular endomorphism ring problem given one endomorphism. IACR Communications in Cryptology (CiC) **2**(1), 6 (2025). https://doi.org/10.62056/akgyivrzn 9, 30
25. Montgomery, P.L.: Speeding the Pollard and elliptic curve methods of factorization. Mathematics of Computation **48**, 243–264 (1987). https://doi.org/10.1090/S0025-5718-1987-0866113-7 11

26. National Institute of Standards and Technology: Post-Quantum Cryptography: Additional Digital Signature Schemes (Aug 2022), https://csrc.nist.gov/projects/pqc-dig-sig 2, 30
27. Onuki, H.: On oriented supersingular elliptic curves. Finite Fields and Their Applications **69**, 101777 (2020), https://arxiv.org/abs/2002.09894 5, 6
28. Ortner, R., Woess, W.: Non-backtracking random walks and cogrowth of graphs. Canadian Journal of Mathematics **59**(4), 828–844 (2007) 38
29. Page, A., Robert, D.: Introducing Clapoti(s): Evaluating the isogeny class group action in polynomial time. Cryptology ePrint Archive, Report 2023/1766 (2023), https://eprint.iacr.org/2023/1766 7
30. Petit, C., Lauter, K.: Hard and easy problems for supersingular isogeny graphs. Cryptology ePrint Archive, Report 2017/962 (2017), https://eprint.iacr.org/2017/962 30
31. Silverman, J.H.: The arithmetic of elliptic curves. No. 106 in Graduate Texts in Mathematics, Springer, 2nd edn. (2009) 36
32. Sutherland, A.V.: Isogeny volcanoes. In: ANTS X. The Open Book Series, vol. 1, pp. 507–530. Mathematical Sciences Publishers (2012), https://arxiv.org/abs/1208.5370 7
33. Ullman, J.D.: Computational Aspects of VLSI. Computer Science Press, Rockville, USA (1984) 4
34. Wesolowski, B.: Orientations and the supersingular endomorphism ring problem. In: Dunkelman, O., Dziembowski, S. (eds.) Advances in Cryptology – EUROCRYPT 2022, Part III. Lecture Notes in Computer Science, vol. 13277, pp. 345–371. Springer, Cham, Switzerland, Trondheim, Norway (May 30 – Jun 3, 2022). https://doi.org/10.1007/978-3-031-07082-2_13 6, 36
35. Wesolowski, B.: The supersingular isogeny path and endomorphism ring problems are equivalent. In: 62nd Annual Symposium on Foundations of Computer Science. pp. 1100–1111. IEEE Computer Society Press, Denver, CO, USA (Feb 7–10, 2022). https://doi.org/10.1109/FOCS52979.2021.00109 6
36. Wesolowski, B.: The supersingular isogeny problem in time and memory $p^{1/3+o(1)}$. Preprint (Jul 2026), https://ia.cr/2026/1486 4

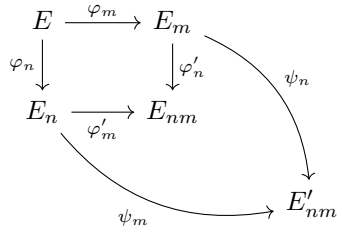
A Lemma on small endomorphisms

Lemma A.1. *Let j, j_n, j_m be pairwise distinct j-invariants such that j_ℓ is ℓ-isogenous to j. Then*

$$\#\{(j, j_n, j_m) \mid \deg(g(j, j_n, j_m)) \geq 2\} \in O((nm)^3).$$

That is, for uniformly distributed j, $g(j, j_n, j_m)$ is overwhelmingly likely to have exactly one root when $n, m \in o(p)$.

Proof. Let $j(E_{nm}), j(E'_{nm})$ be distinct roots of $g(j, j_n, j_m)$. From the following diagram of cyclic isogenies



we obtain the E_n-endomorphism $\sigma = (\varphi'_m)^\vee \varphi'_n(\psi_n)^\vee \psi_m$. If σ is scalar, it must be $\sigma = [nm]$ because $\deg(\sigma) = (nm)^2$. This implies $[m]\varphi'_n(\psi_n)^\vee = [n]\varphi'_m(\psi_m)^\vee$ and that $\varphi'_n(\psi_n)^\vee = [n]\theta$ for some isomorphism $\theta\colon E'_{nm} \to E_{nm}$, a contradiction to $j(E_{nm}), j(E_{nm})$ being distinct. As such, $g(j, j_n, j_m)$ having at least two distinct roots implies that E_n has a non-scalar endomorphism of degree $(nm)^2$. The number of such curves is $O((nm)^3)$ [23, Prop. B.3].

Conversely, if $g(j, j_n, j_m)$ has a double-root $j(E_{nm})$, then E_m, E_{nm} have a double edge in the m-isogeny graph, something which occurs at most $2\ell(2\ell - 1)$ times [2].

B Lemmas about $\sqrt{-\ell p}$

Throughout this section, let $\chi^{(p)}$ denote the Galois conjugate of an isogeny χ.

The following lemma presents a generalization to inseparable isogenies of the fact that an endomorphism of degree $< p/4$ on a supersingular curve defined over $\mathbb{F}_p$ that anticommutes with Frobenius must have trace zero. (A similar, although formally a bit weaker, statement appeared in [16, Lemma 4.4].)

Lemma B.1. *Let $E/\mathbb{F}_{p^2}$ be a supersingular elliptic curve and $E \xrightarrow{\pi} E^{(p)} \xrightarrow{\pi'} E$ its p-power Frobenius isogenies. Assume $E(\mathbb{F}_{p^2}) = E[p+1]$ or $E(\mathbb{F}_{p^2}) = E[p-1]$. Given an isogeny $\psi\colon E \to E^{(p)}$ of degree $\ell < p/4$, define $\mu := \pi' \circ \psi \in \mathrm{End}(E)$. Then μ satisfies $\mu^2 = [-\ell p]$.*

Proof. It is clear that $\deg(\mu) = \ell p$, so we only have to prove $\operatorname{tr}(\mu) = 0$. First, from the assumption $E(\mathbb{F}_{p^2}) = E[p \pm 1]$, note that $\pi' \circ \pi = [\mp p]$, hence $\pi' = \mp\pi^\vee$. Then

$$\operatorname{tr}(\mu) = \pi'\psi + \psi^\vee\pi'^\vee = \pi'\psi \mp \psi^\vee\pi = \pi'\psi \mp \pi'(\psi^\vee)^{(p)} = \pi' \circ (\psi \mp (\psi^\vee)^{(p)}) \,.$$

Write $\psi' = (\psi^\vee)^{(p)}$. Since $\operatorname{tr}(\mu) \in \mathbb{Z}$, we must have $p \mid \deg(\psi \mp \psi')$. However, the Cauchy–Schwarz inequality [31, Lemma V.1.2] shows

$$\deg(\psi \mp \psi') \leq \deg\psi + \deg\psi' + 2\sqrt{\deg\psi \deg\psi'} = \ell + \ell + 2\sqrt{\ell^2} = 4\ell \,.$$

If $\deg(\psi \mp \psi')$ is nonzero, its divisibility by p implies $p \leq \deg(\psi \mp \psi') \leq 4\ell$, which blatantly violates the assumption $\ell < p/4$. Thus, only the case $\deg(\psi \mp \psi') = 0$ is possible, which implies $\psi \mp \psi' = 0$ and therefore $\operatorname{tr}(\mu) = 0$. □

Remark B.2. There are counterexamples to this statement when $\ell \geq p/4$.

Remark B.3. The converse holds unconditionally (for $p \nmid \ell$): An orientation by $\sqrt{-\ell p}$ induces an ℓ-isogeny to the Galois conjugate. This is because all p-isogenies are purely inseparable in the supersingular case, which means any endomorphism of degree ℓp must decompose as an ℓ-isogeny followed by Frobenius.

In the paper we've seen how to reduce Problem 2.2 to Problem 2.4 over $\mathbb{F}_{p^2}$: Given a small degree ℓ, we need to construct a curve with known endomorphism ring that carries an orientation by $\mathbb{Z}[\sqrt{-\ell}]$ or $\mathbb{Z}[\sqrt{-\ell p}]$. Bröker's algorithm handles the case of $\mathbb{Z}[\sqrt{-\ell}]$ and outputs an $\mathbb{F}_p$-curve with an orientation by $\mathbb{Z}[\sqrt{-\ell}]$, and the full endomorphism ring of such a curve can easily be computed from this data. The following lemma shows that the same special curve also carries an orientation by $\mathbb{Z}[\sqrt{-\ell p}]$. (For small ℓ, this approach is both simpler and faster than the more general construction of [34, Lemma 4].)

Lemma B.4. *Let $p \geq 5$ be a prime and let E be a supersingular elliptic curve defined over $\mathbb{F}_p$ with Frobenius endomorphism π. Consider a $\psi \in \operatorname{End}(E)$ such that $\psi \circ \pi = -\pi \circ \psi$ and write $\ell = \deg\psi$. Then $\varphi := \pi \circ \psi$ satisfies $\varphi^2 = [-\ell p]$.*

Proof. First, note that $\operatorname{tr}(\psi) = 0$: Indeed, using the fact that $\pi^2 = [-p]$, we get

$$\pi \circ [\operatorname{tr}(\psi)] = \pi\psi + \pi\psi^\vee = -\psi\pi - \pi^\vee\psi^\vee = [-\operatorname{tr}(\psi\pi)] \,.$$

Since $\pi \notin \mathbb{Z}$, this equality can only hold if $\operatorname{tr}(\psi) = \operatorname{tr}(\psi\pi) = 0$. Therefore we get $\psi^2 = [-\ell]$, which implies

$$\varphi^2 = (\pi\psi)(\pi\psi) = \pi(\psi\pi)\psi = \pi(-\pi\psi)\psi = -\pi^2\psi^2 = -[-p][-\ell] = [-\ell p] \,.$$

Alternative proof: A quaternion order containing $\mathbf{i}$ and $\mathbf{j}$ also contains $\mathbf{k}$. □

The following lemma shows the fact that curves that simultaneously admit two distinct orientations by orders of the form $\mathbb{Z}[\sqrt{-d_i p}]$ occur very rarely in the supersingular isogeny graph.

Lemma B.5. *Let p be a prime number and d_1, d_2 distinct positive integers with $d_i < p/4$. Let $\mathcal{S}_{p^2}$ be the set of supersingular j-invariants over $\mathbb{F}_{p^2}$. We have*

$$\#\{j \in \mathcal{S}_{p^2} \mid j \text{ admits an orientation by } \mathbb{Z}[\sqrt{-d_i p}] \text{ for both } i = 1, 2\} \in \widetilde{O}(d_1 d_2).$$

Proof. By Remark B.3, there are two non-equivalent isogenies $\psi_i \circ E \to E^{(p)}$ of degree d_i, therefore E has an endomorphism of degree $d_1 d_2$. Thus, the j-invariant $j(E)$ must be a root of the univariate polynomial $f(x) = \Phi_{d_1 d_2}(x, x)$, where Φ_ℓ is the ℓ-modular polynomial. The conclusion follows by considering the degree of the modular polynomial.

C Instantiation of small-degree modular polynomials

Algorithm 4 Optimized computation of $\mathcal{N}_2(j_1, j_2)$.

$t_1 \leftarrow 1488 \cdot j_1$ ▷ $2\mathsf{M}_p$
$a_1 \leftarrow -j_1^2 + t_1 - 162000 + j_2$ ▷ $1\mathsf{a}_p + 2\mathsf{a}_{p^2} + 1\mathsf{S}_{p^2}$
$a_2 \leftarrow j_1(t_1 + 40773375) + 8748000000 + a_1 j_2$ ▷ $2\mathsf{a}_p + 1\mathsf{a}_{p^2} + 1\mathsf{S}_{p^2} + 1\mathsf{M}_{p^2}$
return $x^2 + a_1 x + a_2$ ▷ Total: $3\mathsf{a}_p + 2\mathsf{M}_p + 3\mathsf{a}_{p^2} + 1\mathsf{S}_{p^2} + 2\mathsf{M}_{p^2}$

Algorithm 5 Optimized computation of $\mathcal{N}_3(j_1, j_2)$.

$t_1 \leftarrow 8900222976000 \cdot j_1$ ▷ $2\mathsf{M}_p$
$t_2 \leftarrow 2232 \cdot j_1$ ▷ $2\mathsf{M}_p$
$t_3 \leftarrow j_1^2$ ▷ $1\mathsf{S}_{p^2}$
$c_1 \leftarrow -770845966336000000$
$c_2 \leftarrow 1855425871872000000000$
$a_1 \leftarrow j_1(-t_3 + t_2 - 1069956) + 36864000 + j_2$ ▷ $2\mathsf{a}_p + 2\mathsf{a}_{p^2} + 1\mathsf{M}_{p^2}$
$a_2 \leftarrow t_3(t_2 + 2587918086) + t_1 + 452984832000000 + a_1 j_2$ ▷ $2\mathsf{a}_p + 2\mathsf{a}_{p^2} + 2\mathsf{M}_{p^2}$
$a_3 \leftarrow j_1(-1069956 \cdot t_3 + t_1 + c_1) + c_2 + a_2 j_2$ ▷ $2\mathsf{a}_p + 2\mathsf{a}_{p^2} + 2\mathsf{M}_p + 2\mathsf{M}_{p^2}$
return $x^3 + a_1 x^2 + a_2 x + a_3$ ▷ Total: $6\mathsf{a}_p + 6\mathsf{M}_p + 6\mathsf{a}_{p^2} + 1\mathsf{S}_{p^2} + 5\mathsf{M}_{p^2}$

D More precise probability analysis of linear walks

In this section we provide another proof of Lemma 3.6, where we argue more precisely about the big-O approximation terms in the success probability of a linear-walk search for the $\mathbb{F}_p$-subgraph.

Proof. Write $r_{\mathcal{O}} = r_{\ell,\mathcal{O}}$, $\mathcal{E}(\mathcal{O}) = \mathcal{E}_\ell(\mathcal{O})$, and $f_\ell = f_{\ell,\mathcal{O}}$, for $\mathcal{O} = \mathbb{Z}[\sqrt{-p}]$. Let $P_n := \mathbb{P}(\forall i \leq n\ E_i \notin \mathcal{S}_p)$ be the complementary probability of the one we want to

estimate. P_{n+1} with P the probability that the endpoint of the path is adjacent to the $\mathbb{F}_p$ subgraph and that the last step is the ascending isogeny towards $\mathbb{F}_p$. As there are approximately $f_\ell \cdot \#\mathcal{E}(\mathcal{O})$ nodes adjacent to the $\mathbb{F}_p$-subgraph and (with $O(1)$ exceptions) only one of the isogenies leads to the $\mathbb{F}_p$ subgraph

For $n = 1$, the non-backtracking walk is just a uniformly random supersingular elliptic curve, thus we can see $P_1 = 1 - \#\mathcal{S}_p/\#\mathcal{S}_{p^2} = 1 - r_p$. The conclusion follows by induction if we prove $P_{n+1} = P_n \cdot (1 - f_\ell/(\ell+1)\, r_\mathcal{O} + O(n\, r_\mathcal{O}^2))$.

The latter statement follows from heuristic graph-theoretic arguments on non-backtracking random walks.

The probability P_n is the ratio between the number of length-n directed non-backtracking walks on $\mathcal{G}$ and the number of length-n directed non-backtracking walks in $\mathcal{G}_\ell$. As $\mathcal{G}_\ell$ is $(\ell+1)$-regular, the latter is $(\ell+1) \cdot \ell^{n-1} \cdot \#\mathcal{S}_{p^2}$.

The former quantity can be estimated using the Hashimoto matrix (non-backtracking matrix) of $\mathcal{G}_\ell(\mathbb{F}_{p^2})$ (see [28] for details). Let A_{p^2} be this matrix and e the vector with all components equal to 1. As $\mathcal{G}_{p^2}$ is an expander graph, we have that $A_{p^2}^n v$ rapidly converges to e for all v. The matrix A counting non-backtracking paths in $\mathcal{G}$ is very close to ℓA_{p^2}; more precisely, the row corresponding to an edge v_i is the same in A and ℓA_{p^2} whenever the end vertex of an edge is not adjacent to $\mathbb{F}_p$.

If we let e be the vector with all components equal to 1, and $r = \ell e - Ae$ be the vector representing the "missed" edges that would lead to the $\mathbb{F}_p$ subgraph. Note that the nonzero components of r are at most $f_\ell \#\mathcal{E}(\mathcal{O})$ (corresponding to the edges ending in a vertex adjacent to $\mathbb{F}_p$). First note that $e^\top e = ||e||_1$ is the number $E = (\ell+1)\#\mathcal{S}_{p^2} - \ell f_\ell \#\mathcal{E}(\mathcal{O})$ of directed edges in G, while $e^\top r$ is the number of directed edges whose end node is adjacent to $\mathbb{F}_p$. By the structure of $\mathcal{G}_{\mathcal{O},\ell}$ we have $\frac{e^\top r}{e^\top e} = r_\mathcal{O} \cdot \ell f_\ell / \ell + 1$.

The number of non-backtracking walks of length n is $e^\top A^n e$, that is,

$$e^\top (\ell^n e + \sum_{i=0}^{n-1} \ell^{n-1-i} A^i r)$$

So we have

$$\frac{P_{n+1}}{P_n} - 1 = \frac{e^\top A^{n+1} e}{(\ell+1)\cdot \ell^n \#\mathcal{S}_{p^2}} \frac{(\ell+1)\cdot \ell^{n-1}\#\mathcal{S}_{p^2}}{e^\top A^n e} - 1 = \frac{e^\top A^n (Ae - \ell e)}{\ell e^\top A^n e} = -\frac{e^\top A^n r}{\ell e^\top A^n e}$$

Similarly, we have $e^\top A \approx \ell e - r^\top A$: indeed, for each of the edges v_i whose end node is adjacent to $\mathbb{F}_p$ (for these i, $r_i \neq 0$, or better $r_i = 1$ except in a negligible fraction of the cases), the possible next edges in $\mathcal{G}$ (corresponding to the nonzero components of $r^\top A$) miss one ingoing edge. We now compute

$$\frac{e^\top A^n r}{\ell e^\top A^n e} = \frac{(\ell^n e^\top + r^\top \sum_{i=1}^{n} \ell^{n-i} A^i) r}{\ell^{n+1} e^\top e + e^\top \sum_{i=0}^{n-1} \ell^{n-1-i} A^i r}$$

The expressions of the form $r^\top A^i r / e^\top A^i e$ are instead in $O(r_\mathcal{O}^2)$: indeed, this is the portion of nodes that starts and end in one of the edges $r_i \neq 0$: Such paths

are $O(1)$ for small i, and by the rapid mixing properties of the supersingular isogeny graph, (as the endpoint of a path tends to be uniformly distributed) $r^\top A^i r$ tends to be a fraction α of all the $r^\top A^i e$ paths starting from nodes in r, with $\alpha = ||r||_1/||e||_1 \in O(r_\mathcal{O})$. Expanding the expressions further, we notice then that all terms of the form $e^\top A^i r/e^\top A^i e$ are in $O(r_\mathcal{O})$.

By expanding the above expression in terms of powers of $r_\mathcal{O}$, using $(e^\top e + e^\top \sum_i \ell^{n-1-i} A^i r)^{-1} = (e^\top e - e^\top \sum_i \ell^{n-1-i} A^i r)/e^\top e$, we note the leading term of $(e^\top A^n r/(\ell e^\top A^n e)$ is $e^\top r/(\ell e^\top e) = f_\ell/(\ell+1)r_\mathcal{O}$, followed by $O(n)$ terms lying in $O(r_\mathcal{O}^2)$.

This shows $P_{n+1}/P_n - 1 = f_\ell/(\ell+1)r_\mathcal{O} + O(nr_\mathcal{O}^2)$ as desired.

E Exploring along a tree

$\mathsf{Searcher}_2$ (resp. $\mathsf{SuperSearcher}_2$) [9] tests for $\mathbb{F}_p$-ness (resp. performs the same tests as SuperSolver) on each visited node, but explores nodes in a 2-isogeny tree of height h centered at a random starting curve, instead of walking linearly. We analyze here some consequences on the expected number of visited nodes, compare our predictions with the theoretical estimates of [9] and run experiments to gather relevant data on the matter.[18]

The algorithms in [9] are applied to specific prime characteristics: namely, HD-friendly primes as the ones used in SQIsign, satisfying $p \equiv 7 \pmod 8$: in the following, we assume given such a prime p.

Let $\mathcal{O} = \mathbb{Z}[\sqrt{-p}]$ and $\#\mathcal{E}(\mathcal{O})$ be the number of floor $\mathbb{F}_p$-nodes in $\mathcal{G}_2(\mathbb{F}_{p^2})$. By the local tree-like structure of isogeny graphs, there are at most $\#\mathcal{E}(\mathcal{O}) \cdot 2^d$ nodes at distance d from $\mathcal{G}_2(\mathbb{F}_p)$, and $\#\mathcal{E}(\mathcal{O}) \cdot 2^{d+1}$ at distance *at most* d.

To make the probability of hitting an $\mathbb{F}_p$-node (resp., a distinguished node) in the tree reasonably close to 1, we need $\#\mathcal{E}(\mathcal{O}) \cdot 2^{d_0+1} \approx \#\mathcal{S}_{p^2}$ (resp. $\#\mathcal{E}(\mathcal{O}) \cdot 2^{d_0+1} \cdot (1 + \sum'_\ell f'_\ell) \approx \#\mathcal{S}_{p^2}$). A tree of height d contains approximately $3 \cdot 2^d$ nodes; by the relation $\#\mathcal{E}(\mathcal{O}) = \#\mathcal{S}_p/2$ that holds in the considered prime characteristics, in the case of $\mathrm{Searcher}_2$ we have $d \approx \log_2(\mathcal{S}_{p^2}/\mathcal{S}_p)$: the expected number of visited nodes is $3 \cdot \mathcal{S}_{p^2}/\mathcal{S}_p$.

It is noted in [9] that, for a given d, we're not sure to find a special node in the tree, given that the tree is centered in a random node, explored depth-first starting from a random branch and the special node is itself randomly located. the author takes this failure probability into account, but does not consider the fact that the visited nodes are (not independently random) nodes of a tree, nor the discrepancy between the number of $\mathbb{F}_p$-nodes and floor nodes. Therefore, the cost claimed in [9] is underestimated by a factor between 2 and 3.

Remark E.1. For comparison, in the setting of Lemma 3.6 (a linear walk with $\ell = 2$ and no neighbourhood detection) the expected number of visited nodes is the same: $3 \cdot \mathcal{S}_{p^2}/\mathcal{S}_p$. Indeed, the probability of hitting $\mathbb{F}_p$ in exactly n steps essentially follows a geometric distribution with parameter $2/3 \cdot r_\mathcal{O} = r_p/3$.

[18] See `estimator/walk_tree.py`.

Remark E.2. If d is increased from d_0 to $d_0 + h$, the cost of exploring a full tree increases by a multiplicative factor 2^h but the expected number of special nodes in the tree (inversely proportional to the expected number of visited nodes) increases too. If d is decreased to $d_0 - h$, the cost of exploring a full tree decreases by a multiplicative factor 2^h, but with probability $1 - 2^{-h}$ a full tree yields no solution and we are forced to explore more (random) trees (the number of trees we need follows a geometric distribution) until we find a special node. In both cases, experiments[19] suggest that the expected cost stays constant as d varies.

In [9], variants of $\mathsf{SuperSearcher}_2$ using 4- and 12-isogeny trees are proposed. These perform better than the variant with 2-isogenies, but are more complex to analyze. For a more thorough theoretical framework analyzing depth-first tree exploration in isogeny graphs, we refer the reader to [11].

F Negative results

In this section, we review a number of ways one can tweak the $(2, 3)$-isogeny ladders that we used in the paper to explore isogeny graphs, and argue why these tweaks turn out to be essentially ineffective.

Using isogenies instead of modular polynomials. Chi-Dominguez's work [9] improves upon the cost of SuperSolver [10] for specific characteristics, by using $\mathbb{F}_{p^2}$-rational 2^n-torsion points and 3-radical isogenies.

One might try to apply these ideas to Algorithm 2 to optimize our ladder explorations by using rational 2- and 3-isogenies instead of modular polynomials. However, this does not lead to improvements.

Indeed, we analyze here the cost of computing a *pushout* as in Section 3.1. For a given curve E_t^d in the ladder (see Figure 1), we should store a triple (A_t^d, P_t^d, Q_t^d) consisting of the Montgomery coefficient of the curve E_t^d and torsion points $P_t^d \in E_t^d[2]$ and $Q_t^d \in E_t^d[3]$. Given data for E_{t-1}^{d-1} and E_t^{d-1}, retrieving the data relative to the pushout curve E_t^d requires computing the Montgomery coefficient A_t^d of the codomain of a 2-isogeny, evaluating a 2-isogeny and a 3-isogeny (with kernel generated by P_t^{d-1}, $Q^{d-1})_{t-1}$ respectively) and finally retrieving the j-invariant from A_t^d. To optimize the cost (at the expense of memory usage) we can perform all computations projectively (including the check of definedness over $\mathbb{F}_p$).

Exploiting the symmetry of the isogeny formulas in the Montgomery model, we can retrieve the extra 2-neighbour $\mathcal{T}_t^d$ of E_t^{d-1} (see notations of Figure 3). Given the obtained nodes, we can check whether both are in $\mathbb{F}_p$, or whether E_t^d is oriented by $\mathbb{Z}[\sqrt{-2p}]$ by comparison with its 2-neighbours.

Following the costs referenced in [9], we have:

1. Finding the codomain costs $2\mathsf{S}_{p^2} + 1\mathsf{a}_{p^2}$

[19] See `estimator/walk_tree.py`: we tested tree heights between $d_0 - 5$ and $d_0 + 8$, with $d_0 = \log_2 r_{2,\mathbb{Z}[\sqrt{-p}]}^{-1}$ for Searcher and $d_0 = \log_2 r_{2,\mathbb{Z}[\sqrt{-p}]}^{-1}/(1 + \sum_\ell f_{\ell,\mathbb{Z}[\sqrt{-p}]})$ for SuperSearcher, as in the discussion above.

2. Evaluating a 3-isogeny costs $4\mathsf{M}_{p^2} + 2\mathsf{S}_{p^2} + 4\mathsf{a}_{p^2}$
3. Evaluating a 2-isogeny costs $4\mathsf{M}_{p^2} + 6\mathsf{a}_{p^2}$.
4. Computing a j-invariant costs $2\mathsf{M}_{p^2} + 4\mathsf{S}_{p^2} + 8\mathsf{a}_{p^2}$ each.
5. Checking whether the ratio u/v is in $\mathbb{F}_p$, for $u, v \in \mathbb{F}_{p^2}$, costs $2\mathsf{M}_p$.
6. Checking whether two ratios u_1/v_1 and u_2/v_2 are Galois conjugates amounts to comparing the products $u_1\overline{v_2}$ and $u_2\overline{v_1}$. Most of the time we can just check equality of the $\mathbb{F}_p$-component (the "real parts"), so we can use $4\mathsf{M}_p$ per comparison.

The total number of multiplications is

$$12\mathsf{S}_{p^2} + 12\mathsf{M}_{p^2} + 12\mathsf{M}_p = 72\mathsf{M}_p$$

which is larger than the cost $41\mathsf{M}_p$ of a single pushout in Algorithm 2, and does not easily allow for the inspection of the other 3-neighbours, therefore providing a lower success probability than our algorithm.

Surprisingly, a large part of the cost is given by the computation of j-invariants. Moreover, as the j-invariants are computed projectively, performing oriented neighbour detection becomes much more expensive than in the affine case. Either inverting a j-invariant or evaluating an ℓ-modular polynomial at it projectively, for an $\ell \in \{2, 3\}$, exceeds the cost of 50 multiplications, hence does not improve the overall cost either.

Further neighbourhood inspection. Running neighbourhood inspection (see [10] and Section 2.4) with primes $\ell > 3$ is not rewarding either.[20]

The cost c_ℓ of inspection around a given node for a given prime ℓ equals at least the cost of instantiating the ℓ-th modular polynomial summed together with the cost of an inversion-free gcd. As $\ell > 3$, we cannot use non-backtracking modular polynomials, so we have $c_\ell \geq 4(\ell+1)^2 - 9\mathsf{M}_p$, and the added summand in the success probability factor is $z_\ell = f_{2,\mathcal{O}} f_{3,\mathcal{O}} f_{\ell,\mathcal{O}} r_{2,3,\mathcal{O}}/12$ (where we used $f_{2,\mathcal{O}} = 2, f_{\ell,\mathcal{O}} \leq \ell+1$). Performing an extra test is convenient (*i.e.* decreases the overall cost) if and only if the ratio $R = c_\ell/z_\ell$ between the cost and the success probability of the test is lower than the corresponding ratio R' of the algorithm without the test; in this case, we always have $R > R'$. In Algorithm 2, ignoring orientability detection, the cost per visited node is of $85\mathsf{M}_p$ with a success probability factor of $f_{2,\mathcal{O}} f_{3,\mathcal{O}} f_{\ell,\mathcal{O}} r_{2,3,\mathcal{O}} \cdot 7/12$. We see that ℓ-neighbourhood inspection is not convenient observing that $R/R' = \frac{4(\ell+1)^2-9}{f_{\ell,\mathcal{O}}} \frac{7}{85} \geq \frac{4(\ell+1)^2-9}{\ell+1} 7/85$ is always larger than 1.

Further orientability detection. Consider now orientability detection by a prime $\ell \geq 5$. As above, assume we'd have to evaluate the ℓ-th modular polynomial. On its own, the cost is $(3\ell^2+\ell+3)\mathsf{M}_p$ [11, Lemma 1], and the success ratio $f_2 f_3/12 \cdot r_\ell$. Approximating r_ℓ with $\alpha \cdot r_p$ for some $\alpha \gtrsim 1$, and $f_2 = 2$, $f_\ell \approx \ell+1$, the resulting cost-per-node ratio is expected to be around $3/2 \cdot (3\ell^2 + \ell + 3)\alpha^{-1} r_p^{-1}$. This is only helpful in the rare cases when r_ℓ/r_p is sufficiently large; experimentally, for

[20] For experimental verification of this claim, see `estimator/hypersolver_vs_supersolver.py`.

most primes this is not the case. Moreover, further orientability detection would complicate the code base and introduce memory management issues, which are detrimental for our goal of an optimized implementation for GPU (which is by design memory-contrained), therefore we don't consider this as a viable optimization of our algorithm. Note, however, that ℓ-orientability detection shares considerable computation with ℓ-neighbourhood inspection; this could lead to further speedup in the aforementioned rare cases.

Mixing ladders and trees. As seen in Appendix E, exploring a tree does not improve the expected number of visited nodes when compared to a linear walk, but lowers the cost per visited node. One might thus think to replace the outer 2-wall with a 2-isogeny tree. This indeed halves the cost of generating the outer wall: every square root we compute gives two nodes instead of one. However, in our algorithm, the cost of the outer walls is already amortized out and is almost irrelevant in the final complexity; on the other hand, the cost per pushout (dominating in the complexity of Algorithm 2) would be left the same. That is, this optimization would yield negligible performance gains while, as above, complicating the implementation and introducing memory management problems.

Multi-dimensional $(\ell_1, \ell_2, \ldots, \ell_m)$-ladders. Consider extruding the $(2, 3)$-isogeny ladder Figure 1 in one extra dimension, via the use of an extra prime $\ell_3 > 3$. As in the approach above, this would help amortize the cost of the outer sides of the ladder. On the other hand, using larger primes make pushouts more expensive, thus increasing both the time complexity and space complexity of our algorithm.