Improved Subfield Curve Search For Specific Field Characteristics

Jesús-Javier Chi-Domínguez

Cryptography Research Center, Technology Innovation Institute, Abu Dhabi, UAE

Abstract. Isogeny-based cryptography relies its security on the hardness of the supersingular isogeny problem: finding an isogeny between two supersingular curves defined over a quadratic field extension of $\mathbb{F}_p$.
The Delfs-Galbraith algorithm is one of the most efficient procedures for solving the supersingular isogeny problem with a time complexity of $\tilde{\mathcal{O}}(p^{1/2})$ operations. The bottleneck of the Delfs-Galbraith algorithm is the so-called subfield curve search (i.e., finding an isogenous supersingular elliptic curve defined over the prime field), which determines the time complexity of the aforementioned algorithm.
Given that, for efficiency, most recent isogeny-based constructions propose using finite fields with field characteristics equal to $p = 2^a \cdot f - 1$ for some positive integers a and f. This work primarily focuses on primes of that particular form and presents two heuristic algorithms for finding subfield curves with a time complexity of $\mathcal{O}(p^{1/2})$ operations and a memory complexity polynomial in $\log_2 p$. We show how to adapt these algorithms to primes of the form $p = d^a \cdot f - 1$ with d being a small integer, with a particular focus on $d = 12$. We provide concrete time-complexity bounds for both kinds of primes $p = 2^a \cdot f - 1$ and $p = 12^a \cdot f - 1$. Our algorithms exploit the existence of large d^a-torsion points and extend the subfield root detection algorithm of Corte-Real Santos, Costello, and Shi (Crypto, 2022) to a projective scenario where one can work with the curve coefficients instead of their explicit j-invariants.
We highlight that our algorithms easily extend to primes of the form $p = 2^a \cdot f + 1$, $p = 4 \cdot d_1 \cdot d_2 \cdots d_n - 1$ for small odd primes d_i's, and primes such that $p - 1$ and $p + 1$ are B-smooth for some integer B.

Keywords: Cryptanalysis · Delfs-Galbraith Algorithm · Isogeny-based Cryptography · Subfield Curve Search · Supersingular Isogeny Problem

1 Introduction

The security of most of the isogeny-based cryptographic schemes relies on the hardness of the *supersingular isogeny problem*, which refers to finding an isogeny $\phi\colon \mathcal{E} \to \mathcal{F}$ between two given supersingular elliptic curves defined over $\mathbb{F}_{p^2}$.

The best classical algorithm for solving the supersingular isogeny problem is the Delfs-Galbraith algorithm [DG16], which splits into two main steps:

1. *Subfield curve search*: Fix an integer ℓ, and find two $\ell^\bullet$-isogenies, $\phi_0\colon \mathcal{E} \to \mathcal{E}'$ and $\phi_1\colon \mathcal{F} \to \mathcal{F}'$, so the codomain curves $\mathcal{E}'$ and $\mathcal{F}'$ are defined over the subfield $\mathbb{F}_p$. [1] This step runs in $\tilde{\mathcal{O}}(p^{1/2})$ field operations.
2. *Subfield isogeny search*: Find an isogeny $\psi\colon \mathcal{E}' \to \mathcal{F}'$ between $\mathcal{E}$ and $\mathcal{F}'$ defined over $\mathbb{F}_p$. This step runs in $\tilde{\mathcal{O}}(p^{1/4})$ field operations.

E-mail: jesus.dominguez@tii.ae (Jesús-Javier Chi-Domínguez)

[1] The notation "$\ell^\bullet$-isogeny" refers to a ℓ^e-isogeny for some integer number e.

This work is licensed under a "CC BY 4.0" license.
Date of this document: 2026-07-15.

The Delfs-Galbraith algorithm has a running time of $\tilde{\mathcal{O}}(p^{1/2})$ field operations, and its bottleneck step is the subfield curve search. Hence, any improvement in the subfield curve search of the Delfs-Galbraith algorithm would impact the security of most isogeny-based constructions.

Motivation. Given that the shape of the field characteristic p impacts the field arithmetic and performance of isogeny calculations, and it does not affect security, recent works push for primes of the form $p = 2^a \cdot f - 1$. For example,

- SQIsign v1.0 [CSCD+23] requires primes $p = (4 \cdot 3)^a \cdot f - 1$ with $f \in \mathbb{N}$ and $a \approx \frac{1}{2} \log_{12} p$.
- SQIsignHD [DLRW24] suggests primes $p = 2^a \cdot 3^b \cdot f - 1$ with small $f \in \mathbb{N}$, $a \approx \frac{1}{2} \log_2 p$, and $b \approx \frac{1}{2} \log_3 p$.
- ApresSQI [SEMR24] works over primes of the form $p = 2^a \cdot f - 1$ with $f \in \mathbb{N}$ and $a \approx \frac{1}{2} \log_2 p$.
- SQIsign2D-West [BDD+24], SQIPrime [DF24], SQIsign2D-East [NOC+24], and SQIsign v2.0 [AAA+25] ask for $p = 2^a \cdot f - 1$ with small $f \in \mathbb{N}$ and $a \approx \log_2 p$.

All the above prime numbers determine an interesting structure in the $(p+1)$-torsion subgroup of the curves: there are points of large torsion 2^a. To our knowledge, the Delfs-Galbraith algorithm does not exploit the structure of the torsion subgroup of the curve; it performs isogeny walks through modular polynomials at the cost of field exponentiations.

Our Contribution. We show that the existence of a large degree 2^a-torsion subgroup reduces the complexity of the Delfs-Galbraith algorithm from $\tilde{\mathcal{O}}(p^{1/2})$ to $\mathcal{O}(p^{1/2})$ field operations.

Firstly, we propose a dedicated and optimized subfield search through a Depth-First-Search traversal of the 2-isogeny tree over specific primes, $p = 2^a \cdot f - 1$. Our algorithm performs the subfield search by calculating 2-isogenies from kernel generators, and therefore, it operates on the curves instead of the j-invariants. Such an approach does not require the calculation of the expensive square roots as in the original Delfs-Galbraith algorithm. Next, we extend our algorithm to the setting $p = d^a \cdot f - 1$ for some $a, d, f \in \mathbb{N}$, with a particular focus on $d = 12$.

Secondly, we extend the subfield root detection algorithm from [SCS22] to our case study. Given that our proposed algorithm does not operate on the j-invariants of the curves but on the curves themselves, it implies we cannot directly use the subfield root detection algorithm of [SCS22] without calculating a field inversion. However, we show how to extend that subfield root detection algorithm when we only have the denominator and numerator of the j-invariants of the curves (that is, without requiring any field inversion).

In a nutshell, we propose two new algorithms, $\mathsf{Searcher}_d$ and $\mathsf{SuperSearcher}_d$, with a time complexity equal to

$$c \cdot \frac{\#\mathbb{S}_{p^2}}{\#\mathbb{S}_p} = \mathcal{O}\left(p^{1/2}\right)$$

and a memory complexity polynomial in $\log_2 p$, where $\#\mathbb{S}_{p^2} \approx \frac{1}{12} p$ and $\#\mathbb{S}_p = \tilde{\mathcal{O}}(p^{1/2})$ determine the number of supersingular elliptic curves with j-invariants defined over $\mathbb{F}_{p^2}$ and $\mathbb{F}_p$, respectively. We present explicit time complexities regarding field operations through x-only points on the Kummer lines and validate our claims through extensive experiments (see Table 1). We provide a proof-of-concept implementation using SageMath [The25] available at [Com26].

Our results should be taken as asymptotic improvements: in the matter that the bitlength of p increases, the bigger improvements for the subfield curve searching (see Table 4). In particular, our results should be taken as cautionary advice for proposing

Table 1: Here $n = \log_2\left(\frac{\#\mathbb{S}_{p^2}}{\#\mathbb{S}_p}\right) \approx \frac{1}{2}\log_2 p$. Second and Third columns concern to a single execution of each algorithm: they present the success probability ζ in encountering a subfield curve and the time complexity of the algorithm. Last two columns present the average running time and average number of curve inspected for finding a subfield curve when calling $\frac{1}{\zeta}$ times each algorithm with randomized inputs as discussed in Section 4.4. More precisely, column "Avg. Time" presents $(\frac{1}{\zeta} - 1)$ times the value of column "Runtime".

Algorithm	**Complexity per Call**		**Overall Complexity**	
	Prob. ζ	**Runtime**	**Avg. Time**	**Insp. Curves**
$\mathsf{Searcher}_2$	$\gtrsim 0.63$	$66 \cdot 2^n$	$71.76 \cdot 2^n$	$1.09 \cdot 2^n$
$\mathsf{Searcher}_4$	$\gtrsim 0.73$	$82 \cdot 2^n$	$71.32 \cdot 2^n$	$1.16 \cdot 2^n$
$\mathsf{Searcher}_{12}$	$\gtrsim 0.66$	$127 \cdot 2^n$	$128.92 \cdot 2^n$	$1.11 \cdot 2^n$
$\mathsf{SuperSearcher}_2$	$\gtrsim 0.80$	$65 \cdot 2^n$	$48.75 \cdot 2^n$	$1.22 \cdot 2^n$
$\mathsf{SuperSearcher}_4$	$\gtrsim 0.66$	$42.96 \cdot 2^n$	$43.61 \cdot 2^n$	$1.10 \cdot 2^n$
$\mathsf{SuperSearcher}_{12}$	$\gtrsim 0.63$	$39.49 \cdot 2^n$	$42.94 \cdot 2^n$	$1.09 \cdot 2^n$

conservative parameter sets in isogeny-based constructions with 2λ-bits prime numbers of the form $p = 2^a \cdot f - 1$ with $a \geq \lambda$ and $\lambda \in \{128, 192, 256\}$. For instance, most isogeny-based constructions propose parameter sets that take into account the asymptotic bit complexity $\mathcal{O}\left((\log_2 p)^\kappa p^{1/2}\right)$ taking the value of $\kappa = 2$ from [EHL+20]. This consideration is, for example, presented in SQIsign v2.0 [AAA+25, Section 10.2.1] and aligned to the NIST security considerations of [Nat22, Section 4.B.3]. However, our results suggest a smaller value of $\kappa = \log_3 5 \approx 1.465$ compared to the state of the art; in particular, if we consider the Toom-Cook algorithm for the multiplication field elements in $\mathbb{F}_p$, then our results suggest a smaller bit complexity compared to the state of the art. In summary, our algorithms save a factor of $\log_2 p$ compared to the SuperSolver algorithm of [SCS22], and around $\frac{1}{2}\log_2 p$ compared to [EHL+20].

We stress that our results easily extend to more generic primes like $p = 2^a \cdot f + 1$, $p = d^a \cdot f - 1$, $p = 4 \cdot d_1 \cdot d_2 \cdots d_n - 1$ for small odd primes d_i's, and primes such that $p - 1$ and $p + 1$ are B-smooth for some integer B. We provide a small discussion on that families of primes in the Appendix.

Organization. Section 2 provides the preliminaries, and Section 3 includes a high level description of the algorithm SuperSolver. Section 4 describes the two algorithms, $\mathsf{Searcher}_d$ and $\mathsf{SuperSearcher}_d$ with special focus on $d = 2, 4, 12$; in particular, Section 4.3 presents our projective fast subfield root detection. We draw our experiments in Section 5 and discuss in Section 6 some implications of our work to the state-of-the-art constructions.

2 Preliminaries

Through this paper, we denote the multiplications, squares, and additions in $\mathbb{F}_{p^2}$ by M, S, and A, respectively. On the other hand, mul, sqr, and add denote multiplications, squares, and additions in $\mathbb{F}_p$. [2] For any $x \in \mathbb{R}$, we denote the nearest integer to x by $\lfloor x \rceil$.

Let $p = 2^a \cdot f - 1$ be a prime number of r-bits such that $p \equiv 3 \bmod 4$ and $a \geq \frac{1}{2}r$. Let $\mathbb{F}_{p^2} = \mathbb{F}_p[i]/(i^2 + 1)$ be a quadratic field extension of $\mathbb{F}_p$ and $\mathcal{E}$ be supersingular elliptic curves with $\#\mathcal{E}(\mathbb{F}_{p^2}) = (p \pm 1)^2$ points. In particular, we assume $\mathcal{E}$ is given in Montgomery form (i.e, $\mathcal{E}\colon y^2 = x^3 + Ax^2 + x$ for some $A \in \mathbb{F}_{p^2}$). The j-invariant of $\mathcal{E}$ is $j(\mathcal{E}) = 256(A^2 - 3)^3/(A^2 - 4)$, and two elliptic curves $\mathcal{E}$ and $\mathcal{F}$ are isomorphic iff

[2] If $p \equiv 3 \bmod 4$ then we have M = 3mul + 5add and S = 2mul + 3add.

$j(\mathcal{E}) = j(\mathcal{F})$. We use the term "subfield curve" to mean that $j(\mathcal{E})$ belongs to $\mathbb{F}_p$.

An isogeny $\phi\colon \mathcal{E} \to \mathcal{F}$ is a non-constant morphism between two elliptic curves such that $\#\ker\phi < \infty$ and $\phi(\mathbf{0}_\mathcal{E}) = \mathbf{0}_\mathcal{F}$, where $\mathbf{0}_\mathcal{E}$ (resp. $\mathbf{0}_\mathcal{F}$) denotes the point at infinity on $\mathcal{E}$ (resp. $\mathcal{F}$). We say that ϕ is an ℓ-isogeny if $\gcd(\ell, p) = 1$ and $\ker\phi$ has size ℓ. The dual of ϕ is another ℓ-isogeny $\hat{\phi}\colon \mathcal{F} \to \mathcal{E}$ such that $\phi \circ \hat{\phi} = [\ell] = \hat{\phi} \circ \phi$, where $[\ell]\colon P \mapsto [\ell]P$ denotes the multiplication-by-ℓ map. For any prime ℓ, there are exactly $(\ell + 1)$ ℓ-isogenies (up to isomorphism) with domain $\mathcal{F}$, one of them corresponding to $\hat{\phi}\colon \mathcal{F} \to \mathcal{E}$.

We consider the supersingular isogeny graph $\mathcal{X}(\overline{\mathbb{F}}_p, \ell)$ as the directed graph where the vertices are the j-invariants of supersingular elliptic curves defined over $\overline{\mathbb{F}}_p$, and the edges are ℓ-isogenies. However, we only consider curves defined over $\mathbb{F}_{p^2}$ since the j-invariants of supersingular elliptic curves always belong to $\mathbb{F}_{p^2}$. The graph $\mathcal{X}(\overline{\mathbb{F}}_p, \ell)$ is fully connected for every prime ℓ [Koh96], and it is an expander graph [CLG09]. In particular, the set of all j-invariants of supersingular elliptic curves has cardinality $\#\mathbb{S}_{p^2} \approx \frac{1}{12}p$, and only $\#\mathbb{S}_p = \tilde{\mathcal{O}}(p^{1/2})$ of them belong to $\mathbb{F}_p$ [DG16].

Modular polynomials. The ℓth modular polynomial $\Phi_\ell(x, y) \in \mathbb{Z}[x, y]$ is an irreducible symmetric polynomial such that $\Phi_\ell(j(\mathcal{E}), j(\mathcal{F})) = 0$ for any pair of two ℓ-isogenous curves $\mathcal{E}$ and $\mathcal{F}$. If ℓ is a prime number, and $j = j(\mathcal{E})$ is the j-invariant of an elliptic curve $\mathcal{E}$ then $\Phi_\ell(x, j) \in \mathbb{F}_{p^2}[x]$ is a polynomial of degree $d_\ell = (\ell + 1)$.

Kummer line. The Kummer line of $\mathcal{E}$ is $\mathcal{E}/\langle\ominus\rangle \cong \mathbb{P}^1$, where $\ominus\colon P \mapsto -P$ denotes the negation map. A point $\mathrm{x}(P)$ on $\mathcal{E}/\langle\ominus\rangle$ is of the form $(X\colon Z)$ for some $X, Z \in \mathbb{F}_{p^2}$. In particular,

$$\mathrm{x}(P) = \begin{cases} (1\colon 0) & \text{if } P = \mathbf{0}_\mathcal{E} \\ (X\colon Z) & \text{otherwise.} \end{cases}$$

If $\mathrm{x}(P) = (X\colon Z) \neq (1\colon 0)$ then $x_P = X/Z$ determines the x-coordinate of the point P.

Given the three points $\mathrm{x}(P)$, $\mathrm{x}(Q)$, and $\mathrm{x}(P - Q)$ on $\mathcal{E}/\langle\ominus\rangle$, one can efficiently calculate $\mathrm{x}(P + Q)$ and $\mathrm{x}([2]P)$ at the cost of $4\mathsf{M} + 2\mathsf{S} + 6\mathsf{A}$ and $4\mathsf{M} + 2\mathsf{S} + 4\mathsf{A}$ operations, respectively [Mon87]. In addition, computing $\mathrm{x}([3]P)$ requires $7\mathsf{M} + 5\mathsf{S} + 10\mathsf{A}$ [CH17].

Similarly, ℓ-isogenies on $\mathcal{E}$ nicely restrict to $\mathcal{E}/\langle\ominus\rangle$ [CH17, Ren18, BDLS20]; that is, given a point P on $\mathcal{E}$ and an ℓ-isogeny $\phi\colon \mathcal{E} \to \mathcal{F}$ with $\ker\phi = \langle K\rangle$, then $\mathrm{x}(\phi(P))$ only depends on $\mathrm{x}(P)$, $\mathrm{x}([j]K)$ for some integers $j \in [1, \ell] \cap \mathbb{Z}$, and on the Montgomery A-coefficient of $\mathcal{E}$. Table 2 lists the costs for computing codomain curves and pushing points through ℓ-isogenies with $\ell = 2, 3, 4$.

Table 2: The 2-isogeny and 4-isogeny formulas from [Ren18] assume isogeny kernels different from $(0, 0)$. The 3-isogeny cost comes from [CH17].

	2-**isogeny**		4-**isogeny**		3-**isogeny**	
	Codomain	**Evaluate**	**Codomain**	**Evaluate**	**Codomain**	**Evaluate**
Cost:	$2\mathsf{S}$	$4\mathsf{M}$	$4\mathsf{S}$	$6\mathsf{M} + 2\mathsf{S}$	$2\mathsf{M} + 3\mathsf{S}$	$4\mathsf{M} + 2\mathsf{S}$

Through the manuscript, we only consider multiplications and squares in the time complexities of the algorithms, we omit additions and subtractions.

3 On the Delfs-Galbraith Subfield Searching Algorithm

Corte-Real Santos, Costello, and Shi provide an efficient procedure for determining whether an ℓ-isogeny connects a given elliptic curve to a subfield curve [SCS22]. Or, put differently, they describe an efficient algorithm for inspecting all the ℓ-isogenous curves to a given

arbitrary supersingular curve. More precisely, they propose a subfield root detection algorithm NeighbourInFp(ℓ, j), which determines whether the polynomial $\Phi_\ell(x, j)$ has roots in the field $\mathbb{F}_p$; we give a high-level description of NeighbourInFp in Figure 1. The main ingredients of NeighbourInFp include i) an inverse-free GCD procedure and ii) an efficient and explicit description of the polynomials g_1 and g_2, respectively. [3] In addition, they show that the cost of EvaluateModularPolynomial(ℓ, j) is bounded by $9d_\ell(d_\ell - 1)$mul operations, while InverseFreeGCD(g_1, g_2) by

$$\frac{1}{2}(\deg g_1 + \deg g_2 + 2)(\deg g_1 + \deg g_2 + 3) - 6$$

field multiplications in $\mathbb{F}_p$. Hence, NeighbourInFp(ℓ, j) takes $(11d_\ell^2 - 6d_\ell - 5)$mul operations.

InverseFreeGCD(g, h)

1 : Initialize r as $\mathrm{LC}(h) \cdot g$
2 : Initialize s as $\mathrm{LC}(g) \cdot h$
3 : **while** $\deg r > 1$ and $r \neq s$ **do**
4 : $\quad r \leftarrow r - x^{\deg r - \deg s} \cdot s$
5 : $\quad r, s \leftarrow \mathrm{LC}(s) \cdot r, \mathrm{LC}(r) \cdot s$
6 : $\quad$ **if** $\deg r \geq \deg s$
7 : $\quad\quad r, s \leftarrow s, r$
8 : $\quad$ **endif**
9 : **endwhile**
10 : **return** $\neg(\deg r = 1$ and $r \neq s)$

NeighbourInFp(ℓ, j)

1 : $f \leftarrow$ EvaluateModularPolynomial(ℓ, j)
2 : Compute $g_1 = \frac{1}{2}(f + \pi(f))$
3 : Compute $g_2 = -\frac{i}{2}(f - \pi(f))$
4 : **return** InverseFreeGCD(g_1, g_2)

Figure 1: Description of the subfield root detection algorithm from [SCS22]. The inputs ℓ and j of EvaluateModularPolynomial must satisfies that $j(\mathcal{E}) = j$ for some supersingular elliptic curve $\mathcal{E}$ defined over $\mathbb{F}_{p^2}$. The subroutine EvaluateModularPolynomial(ℓ, j) is as described in [SCS22]: it evaluates $\Phi_\ell(x, y)$ at $y = j$ by calculating $j^2, \ldots, j^{d_\ell}$ and multiplying them by the corresponding coefficients of $\Phi_\ell(x, y)$. The inputs g and h of InverseFreeGCD correspond with polynomials $g, h \in \mathbb{F}_{p^2}[x]$ such that $\deg g \geq \deg h$. Here, π denotes the p-th power Frobenius endomorphism on $\mathbb{F}_{p^2}$. [4]

3.1 Overall description of SuperSolver algorithm

As the main application of NeighbourInFp, Corte-Real Santos, Costello, and Shi propose using it in the Delfs-Galbraith Subfield Searching Algorithm. Following the notation in [SCS22], we give in Figure 2 a high-level description of such an optimized subfield-searching algorithm, SuperSolver. To our knowledge, [SCS22] only provides an experimental average cost for SuperSolver of

$$c \cdot \left(\frac{\#\mathbb{S}_{p^2}}{\#\mathbb{S}_p} \right)$$

field operations, where i) $c = 58$ for $\log_2 p \approx 256$, ii) $c = 63.1$ for $\log_2 p \approx 384$, and iii) $c = 69.1$ for $\log_2 p \approx 512$ [SCS22, Table 7]. However, as already highlighted, there is a hidden logarithm factor in the complexity of the Delfs-Galbraith Subfield Searching Algorithm determined by Line 14 in SuperSolver, which requires a square root calculation. Hence, we stress that the time complexity of finding a subfield curve through SuperSolver is $\tilde{\mathcal{O}}(p^{1/2})$ field operations with a memory complexity polynomial in $\log_2 p$.

[3] For instance, we have $\deg g_1 = d_\ell$ and $\deg g_2 = d_\ell - 1$, and the leading coefficient of g_1 is equal to one.

[4] We use the inequality $\deg r > 1$ instead of $\deg r \geq 1$ in Figure 1. In [SCS22, Alg. 1], the authors wrote $\deg r \geq 1$ while in their code they used the inequality $\deg r > 1$.

SuperSolver($\mathcal{E}, h, L$)

1 : Initialize an empty stack for storage of nodes, S
2 : Deterministically compute a 2-isogeny with domain $\mathcal{E}$ and codomain $\mathcal{F}$
3 : Push the root $\mathbf{z}_{\varnothing} = \{j(\mathcal{E}), j(\mathcal{F}), \varnothing\}$ onto S
4 : Pop the first element in S, $\mathbf{z}_{\mathsf{b}} = \{j(\mathcal{E}_{\mathsf{b}}), j(\mathcal{F}_{\mathsf{b}}), \mathsf{b}\}$ where $\mathsf{b} \in \{0,1\}^n$
5 : **if** $j(\mathcal{E}_{\mathsf{b}}) \in \mathbb{F}_p$
6 : **return** $(\mathsf{b}, 2)$
7 : **endif**
8 : **for** each $\ell \in L$
9 : **if** NeighbourInFp$(\ell, j(\mathcal{E}_{\mathsf{b}}))$
10 : **return** (b, ℓ)
11 : **endif**
12 : **endfor**
13 : **if** $n \leq (h-1)$
14 : Compute the two roots $j(\mathcal{E}_{\mathsf{b}0})$ and $j(\mathcal{E}_{\mathsf{b}1})$ of the polynomial $\dfrac{\Phi_2(j(\mathcal{E}_{\mathsf{b}}), x)}{x - j(\mathcal{F}_{\mathsf{b}})}$
15 : Push $\mathbf{z}_{\mathsf{b}1} = \{j(\mathcal{E}_{\mathsf{b}1}), j(\mathcal{E}_{\mathsf{b}}), \mathsf{b}1\}$ onto S
16 : Push $\mathbf{z}_{\mathsf{b}0} = \{j(\mathcal{E}_{\mathsf{b}0}), j(\mathcal{E}_{\mathsf{b}}), \mathsf{b}0\}$ onto S
17 : Go to Line 4
18 : **endif**
19 : **if** S is empty
20 : **return** $\perp$
21 : **else**
22 : Go to Line 4
23 : **endif**

Figure 2: Description of the optimized subfield curve search algorithm from [SCS22]. The input must satisfy $\mathcal{E}$ is a supersingular elliptic curve defined over $\mathbb{F}_{p^2}$, h is a positive integer, and L is a list of positive integers. In practice, L is a list of small (odd prime) numbers. The integer n determines the length of the bitstring b. For the empty bitstring $\mathsf{b} = \varnothing$, we set $n = 0$. In particular, the value of b describes the 2-isogeny path of length n going from $\mathcal{E} = \mathcal{E}_{\varnothing}$ to $\mathcal{E}_n$. For each $i = 0, 1$, the node $\mathbf{z}_{\mathsf{b}i} = \{j(\mathcal{E}_{\mathsf{b}i}), j(\mathcal{F}_{\mathsf{b}i}), \mathsf{b}i\}$ satisfies $\mathcal{E}_{\mathsf{b}i}$ and $\mathcal{F}_{\mathsf{b}i}$ are connected by a 2-isogeny.

4 Subfield Curve Searching Over Specific Fields

This section describes an optimized Depth-First-Search (DFS) traversal of the 2-isogeny trees for finding subfield curves. The algorithm takes inspiration from the DFS technique of [ACC+19].

Let us focus on supersingular elliptic curves defined over $\mathbb{F}_{p^2}$ with $(p+1)^2$ points, which implicitly implies the existence of large 2-torsion points on the curve. Fix a height parameter $h \leq a$, and let $\{P, Q\}$ be a basis of the 2^h-torsion subgroup of a given supersingular elliptic curve $\mathcal{E}\colon y^2 = x^3 + Ax^2 + x$. Next, we inspect the 2-isogeny subtree with root $\mathcal{E}$ and 2^h leaves, excluding the 2-isogeny with kernel generated by $[2^{h-1}]Q = (0,0)$, as described in Figure 3.

The key observation of the above setting is that it allows us to compute 2-isogenies

$\mathsf{Searcher}_2(\mathcal{E}, P, Q, h)$

1 : Initialize an empty stack for storage of nodes, S
2 : Push the root $\mathbf{z}_{\varnothing} = \{\mathcal{E}, \varnothing, P, Q\}$ onto S
3 : Pop the first element in S, $\mathbf{z}_{\mathsf{b}} = \{\mathcal{E}_{\mathsf{b}}, \mathsf{b}, P_{\mathsf{b}}, Q_{\mathsf{b}}\}$ where $\mathsf{b} \in \{0,1\}^n$
4 : **if** $j(\mathcal{E}_{\mathsf{b}}) \in \mathbb{F}_p$
5 : **return** b
6 : **endif**
7 : **if** $n < (h-1)$
8 : Calculate the 2-isogeny $\phi_{\mathsf{b}0} \colon \mathcal{E}_{\mathsf{b}} \to \mathcal{E}_{\mathsf{b}0}$ with $\ker \phi_{\mathsf{b}0} = \langle [2^{h-n-1}]P_{\mathsf{b}} \rangle$
9 : Compute $P_{\mathsf{b}0} = \phi_{\mathsf{b}0}(P_{\mathsf{b}})$ and $Q_{\mathsf{b}0} = \phi_{\mathsf{b}0}([2]Q_{\mathsf{b}})$
10 : Calculate the 2-isogeny $\phi_{\mathsf{b}1} \colon \mathcal{E}_{\mathsf{b}} \to \mathcal{E}_{\mathsf{b}1}$ with $\ker \phi_{\mathsf{b}1} = \langle [2^{h-n-1}](P_{\mathsf{b}} + Q_{\mathsf{b}}) \rangle$
11 : Compute $P_{\mathsf{b}1} = \phi_{\mathsf{b}1}(P_{\mathsf{b}} + Q_{\mathsf{b}})$ and $Q_{\mathsf{b}1} = \phi_{\mathsf{b}1}([2]Q_{\mathsf{b}})$
12 : Push $\mathbf{z}_{\mathsf{b}1} = \{\mathcal{E}_{\mathsf{b}1}, \mathsf{b}1, P_{\mathsf{b}1}, Q_{\mathsf{b}1}\}$ onto S
13 : Push $\mathbf{z}_{\mathsf{b}0} = \{\mathcal{E}_{\mathsf{b}0}, \mathsf{b}0, P_{\mathsf{b}0}, Q_{\mathsf{b}0}\}$ onto S
14 : Go to Line 3
15 : **endif**
16 : **if** $n = (h-1)$
17 : Calculate the 2-isogeny $\phi_{\mathsf{b}0} \colon \mathcal{E}_{\mathsf{b}} \to \mathcal{E}_{\mathsf{b}0}$ with kernel $\ker \phi_{\mathsf{b}0} = \langle P_{\mathsf{b}} \rangle$
18 : Calculate the 2-isogeny $\phi_{\mathsf{b}1} \colon \mathcal{E}_{\mathsf{b}} \to \mathcal{E}_{\mathsf{b}1}$ with kernel $\ker \phi_{\mathsf{b}1} = \langle P_{\mathsf{b}} + Q_{\mathsf{b}} \rangle$
19 : Push $\mathbf{z}_{\mathsf{b}1} = \{\mathcal{E}_{\mathsf{b}1}, \mathsf{b}1, \mathbf{0}_{\mathcal{E}\mathsf{b}1}, \mathbf{0}_{\mathcal{E}\mathsf{b}1}\}$ onto S
20 : Push $\mathbf{z}_{\mathsf{b}0} = \{\mathcal{E}_{\mathsf{b}0}, \mathsf{b}0, \mathbf{0}_{\mathcal{E}\mathsf{b}0}, \mathbf{0}_{\mathcal{E}\mathsf{b}0}\}$ onto S
21 : Go to Line 3
22 : **endif**
23 : **if** S is empty
24 : **return** $\perp$
25 : **else**
26 : Go to Line 3
27 : **endif**

Figure 3: Description of our subfield curve search algorithm, $\mathsf{Searcher}_2$. The input must satisfy $\mathcal{E}$ is a supersingular elliptic curve defined over $\mathbb{F}_{p^2}$, h is a positive integer, and $\mathcal{E}[2^h] = \langle P, Q \rangle$. The integer n determines the length of the bitstring b. For the empty bitstring $\mathsf{b} = \varnothing$, we set $n = 0$. The value of b describes the 2-isogeny path of length n going from $\mathcal{E} = \mathcal{E}_{\varnothing}$ to $\mathcal{E}_n$. For each $i = 0, 1$, the node $\mathbf{z}_{\mathsf{b}i} = \{\mathcal{E}_{\mathsf{b}i}, \mathsf{b}i, P_{\mathsf{b}i}, Q_{\mathsf{b}i}\}$ satisfies $\mathcal{E}_{\mathsf{b}i}[2^{h-n-1}] = \langle P_{\mathsf{b}i}, Q_{\mathsf{b}i} \rangle$ and $\ker \widehat{\phi}_{\mathsf{b}i} = \langle [2^{h-n-2}]Q_{\mathsf{b}i} \rangle$.

through kernel points, and thus, we work over the curves instead of the j-invariants. Therefore, we can avoid calculating square roots when inspecting the 2-isogeny tree rooted at $\mathcal{E}$. We could think that we require a field inversion to compute the j-invariant of each curve in Figure 3, but we highlight that we only need the numerator $(a + ib)$ and the denominator $(c + id)$ of each j-invariant and check if $ad = bc$ holds. That is, we can avoid the field inversions in Figure 3. We state in Proposition 1 the time complexity of $\mathcal{O}(p^{1/2})$ operations for our subfield curve search procedure presented in Figure 3.

Proposition 1. *Let $p = 2^a \cdot f - 1$ be a prime with $a \geq \frac{1}{2}\log_2 p$, and let $h = \log_2\left(\frac{\#\mathbb{S}_{p^2}}{\#\mathbb{S}_p}\right)$.*

The algorithm in Figure 3 has a running time of $\mathcal{O}(2^h)$ field operations over $\mathbb{F}_{p^2}$, and it has a memory complexity polynomial in $\log_2 p$ and h.

Proof. Let us focus on the worst-case scenario (i.e., we inspect all the nodes in the 2-isogeny subtree). Notice that computing the kernel point generators of $\phi_{\mathsf{b}0}$ and $\phi_{\mathsf{b}1}$ can be achieved at the cost of $2 \cdot (h - n - 1)$ point doublings, while the required points in Lines 9 and 11 only need one extra point addition and doubling. Additionally, we need to push four points through 2-isogenies and two 2-isogeny codomain calculations. Thus, inspecting the 2-isogeny subtree requires

- $\sum_{k=0}^{h} 2^k = 2^{h+1} - 1 \approx 2 \cdot 2^h$ subfield checks,
- $\sum_{k=0}^{h-2}(2(h-k-1)+1) \cdot 2^k = 5 \cdot 2^{h-1} - 2h - 3 \approx \frac{5}{2} \cdot 2^h$ point doublings,
- $\sum_{k=0}^{h-1} 2^k = 2^h - 1 \approx 2^h$ point additions,
- $\sum_{k=0}^{h-2} 4 \cdot 2^k = 3 \cdot 2^h - 4 \approx 2 \cdot 2^h$ isogeny evaluations, and
- $\sum_{k=0}^{h-1} 2 \cdot 2^k = 2^{h+1} - 2 \approx 2 \cdot 2^h$ isogeny codomain calculations.

In particular, we get an asymptotic running time of $\mathcal{O}(2^h)$ field operations since each point and isogeny operation takes $\mathcal{O}(1)$ field operations. The memory storage is limited to $(2h + 1)$ stack elements, which gives a memory complexity polynomial in $\log_2 p$ and h. □

Explicit cost on the Kummer line. Given the lesser cost of adding x-only points on the Kummer line than on the elliptic curve, we detailed the running of the proposed method in terms of field operation through x-only points. Thus,

1. The points in $\mathbf{z}_\mathsf{b}$ represent projective x-only points of the form $(X : Z)$.
2. The node $\mathbf{z}_\mathsf{b}$ has an extra point, $\mathbf{z}_\mathsf{b} = \{\mathcal{E}_\mathsf{b}, \mathsf{b}, \mathrm{x}(P_\mathsf{b}), \mathrm{x}(Q_\mathsf{b}), \mathrm{x}(R_\mathsf{b})\}$ where $R_\mathsf{b} = P_\mathsf{b} - Q_\mathsf{b}$.
3. The elliptic curve in $\mathbf{z}_\mathsf{b}$ is represented by its projective representation, $(A' + 2C : 4C)$ where $A = A'/C$.
4. The additional point $\mathrm{x}(R_\mathsf{b})$ implies an extra 2-isogeny evaluation per isogeny, $\mathrm{x}(R_{\mathsf{b}0}) = \mathrm{x}(\phi_{\mathsf{b}0}(P_\mathsf{b} - [2]Q_\mathsf{b}))$ and $\mathrm{x}(R_{\mathsf{b}1}) = \mathrm{x}(\phi_{\mathsf{b}1}(P_\mathsf{b} - Q_\mathsf{b}))$.

We summarize in Lemma 1 the explicit time complexity of the algorithm $\mathsf{Searcher}_2$ instantiated over the Kummer line.

Lemma 1. *Let $p = 2^a \cdot f - 1$ be a prime with $a \geq \log_2\left(\frac{\#\mathbb{S}_{p^2}}{\#\mathbb{S}_p}\right)$, and let $h = \log_2\left(\frac{\#\mathbb{S}_{p^2}}{\#\mathbb{S}_p}\right) - 1$. The algorithm in Figure 3 inspects at most $\frac{\#\mathbb{S}_{p^2}}{\#\mathbb{S}_p}$ curves. In addition, the algorithm instantiated over the Kummer line has a time complexity bounded by*

$$66 \cdot \frac{\#\mathbb{S}_{p^2}}{\#\mathbb{S}_p}\ \mathsf{mul}.$$

Proof. Let us focus on the worst-case scenario (i.e., we inspect all the nodes in the 2-isogeny subtree). This time, inspecting the 2-isogeny subtree performs

- $\sum_{k=0}^{h} 2^k = 2^{h+1} - 1 \approx 2 \cdot 2^h$ subfield checks,
- $\sum_{k=0}^{h-2}(2(h-k-1)+1) \cdot 2^k = 5 \cdot 2^{h-1} - 2h - 3 \approx \frac{5}{2} \cdot 2^h$ point doublings,
- $\sum_{k=0}^{h-2} 2 \cdot 2^k = 2^h - 2 \approx 2^h$ point additions,
- $\sum_{k=0}^{h-2} 6 \cdot 2^k = 3 \cdot 2^h - 6 \approx 3 \cdot 2^h$ isogeny evaluations, and

– $\sum_{k=0}^{h-1} 2 \cdot 2^k = 2^{h+1} - 2 \approx 2 \cdot 2^h$ isogeny codomain calculations.

According to Table 2, and since one can perform a subfield check with $2\mathsf{M} + 4\mathsf{S} + 16\mathsf{A} + 2\mathsf{mul}$ operations, [5] we get a running time bounded by

$$\mathsf{runtime}_2 = (30\mathsf{M} + 19\mathsf{S} + 4\mathsf{mul})2^h = 132 \cdot \frac{\#\mathbb{S}_{p^2}}{\#\mathbb{S}_p}\ \mathsf{mul}.$$

Now, observe that the algorithm $\mathsf{Searcher}_2$ inspects $2^{h+1} - 1$ curves, doubling the required curves $2^h = \frac{\#\mathbb{S}_{p^2}}{\#\mathbb{S}_p}$. Therefore, we can restrict the search to the condition

$$h = \log_2\left(\frac{\#\mathbb{S}_{p^2}}{\#\mathbb{S}_p}\right) - 1.$$

Consequently, the runtime of the algorithm $\mathsf{Searcher}_2$ is bounded by

$$\mathsf{runtime}_2 = 132 \cdot 2^h\ \mathsf{mul} = \frac{132}{2} \cdot \left(\frac{\#\mathbb{S}_{p^2}}{\#\mathbb{S}_p}\right)\mathsf{mul} = 66 \cdot \left(\frac{\#\mathbb{S}_{p^2}}{\#\mathbb{S}_p}\right)\mathsf{mul}. \tag{1}$$

□

4.1 The Impact of Using 4-isogenies instead of 2-isogenies

Computing 4-isogenies is much cheaper than composing two consecutive 2-isogenies (this is already one optimization trick used in many isogeny cryptographic implementations). [6] However, a few differences exist when employing 4-isogenies instead of 2-isogenies in the subfield curve search algorithm. To highlight the usage of 4-isogenies in $\mathsf{Searcher}_2$, we denote the 4-variant algorithm by $\mathsf{Searcher}_4$.

Setup when employing 4-isogenies. Following the notation from Fig. 3, the input must satisfy $\mathcal{E}[4^h] = \langle P, Q\rangle$. In addition, the integer n determines the length of the string $\mathsf{b} \in \{0,1,2,3\}^n$, and we denote $\mathbf{z}_\mathsf{b} = \{\mathcal{E}, \mathsf{b}, \mathrm{x}(P_\mathsf{b}), \mathrm{x}(Q_\mathsf{b}), \mathrm{x}(R_\mathsf{b})\}$ where $R_\mathsf{b} = P_\mathsf{b} - Q_\mathsf{b}$.

Calculating Codomain 4-isogenies when $n < (h-1)$. We need to calculate four different 4-isogenies:

$$\begin{aligned}
&\phi_{\mathsf{b}0}\colon \mathcal{E}_\mathsf{b} \to \mathcal{E}_{\mathsf{b}0} \text{ with } \ker\phi_{\mathsf{b}0} = \langle [4^{h-n-1}](P_\mathsf{b})\rangle,\\
&\phi_{\mathsf{b}1}\colon \mathcal{E}_\mathsf{b} \to \mathcal{E}_{\mathsf{b}1} \text{ with } \ker\phi_{\mathsf{b}1} = \langle [4^{h-n-1}](P_\mathsf{b} + Q_\mathsf{b})\rangle,\\
&\phi_{\mathsf{b}2}\colon \mathcal{E}_\mathsf{b} \to \mathcal{E}_{\mathsf{b}2} \text{ with } \ker\phi_{\mathsf{b}2} = \langle [4^{h-n-1}](P_\mathsf{b} + [2]Q_\mathsf{b})\rangle, \text{ and}\\
&\phi_{\mathsf{b}3}\colon \mathcal{E}_\mathsf{b} \to \mathcal{E}_{\mathsf{b}3} \text{ with } \ker\phi_{\mathsf{b}3} = \langle [4^{h-n-1}](P_\mathsf{b} + [3]Q_\mathsf{b})\rangle.
\end{aligned}$$

Regarding operations in the Kummer line, the cost of computing the above four isogenies includes $3(h-n-1)$ x-only point quadrupling and three x-only point additions.

Calculating Codomain 4-isogenies when $n = (h-1)$. We need to calculate four different 4-isogenies:

$$\begin{aligned}
&\phi_{\mathsf{b}0}\colon \mathcal{E}_\mathsf{b} \to \mathcal{E}_{\mathsf{b}0} \text{ with } \ker\phi_{\mathsf{b}0} = \langle P_\mathsf{b}\rangle,\\
&\phi_{\mathsf{b}1}\colon \mathcal{E}_\mathsf{b} \to \mathcal{E}_{\mathsf{b}1} \text{ with } \ker\phi_{\mathsf{b}1} = \langle P_\mathsf{b} + Q_\mathsf{b}\rangle,\\
&\phi_{\mathsf{b}2}\colon \mathcal{E}_\mathsf{b} \to \mathcal{E}_{\mathsf{b}2} \text{ with } \ker\phi_{\mathsf{b}2} = \langle P_\mathsf{b} + [2]Q_\mathsf{b}\rangle, \text{ and}\\
&\phi_{\mathsf{b}3}\colon \mathcal{E}_\mathsf{b} \to \mathcal{E}_{\mathsf{b}3} \text{ with } \ker\phi_{\mathsf{b}3} = \langle P_\mathsf{b} + [3]Q_\mathsf{b}\rangle = \langle P_\mathsf{b} - Q_\mathsf{b}\rangle.
\end{aligned}$$

Observe that the above kernel calculations add two extra x-only point additions.

[5] Computing the numerator and denominator of $j(\mathcal{E}) = (a+ib)/(c+id)$ from $(A' + 2C\colon 4C)$ requires $2\mathsf{M} + 4\mathsf{S} + 16\mathsf{A}$, and checking $j(\mathcal{E}) \in \mathbb{F}_p$ reduce to verifying $ad = bc$.

[6] We exclude the analysis for employing 8-isogenies since there are no known efficient 8-isogeny formulas better than performing a composition of three 2-isogenies or one 2-isogeny together with a single 4-isogeny.

Pushing Points Through 4-isogenies. We need to push a total of twelve x-only points through the isogenies:

$$\begin{array}{lll}
\mathrm{x}(P_{\mathsf{b}0}) = \mathrm{x}(\phi_{\mathsf{b}0}(P_{\mathsf{b}})), & \mathrm{x}(Q_{\mathsf{b}0}) = \mathrm{x}(\phi_{\mathsf{b}0}([4]Q_{\mathsf{b}})), & \mathrm{x}(R_{\mathsf{b}0}) = \mathrm{x}(\phi_{\mathsf{b}0}(P_{\mathsf{b}} - [4]Q_{\mathsf{b}})) \\
\mathrm{x}(P_{\mathsf{b}1}) = \mathrm{x}(\phi_{\mathsf{b}1}(P_{\mathsf{b}} + Q_{\mathsf{b}})), & \mathrm{x}(Q_{\mathsf{b}1}) = \mathrm{x}(\phi_{\mathsf{b}1}([4]Q_{\mathsf{b}})), & \mathrm{x}(R_{\mathsf{b}1}) = \mathrm{x}(\phi_{\mathsf{b}1}(P_{\mathsf{b}} - [3]Q_{\mathsf{b}})) \\
\mathrm{x}(P_{\mathsf{b}2}) = \mathrm{x}(\phi_{\mathsf{b}2}(P_{\mathsf{b}} + [2]Q_{\mathsf{b}})), & \mathrm{x}(Q_{\mathsf{b}2}) = \mathrm{x}(\phi_{\mathsf{b}2}([4]Q_{\mathsf{b}})), & \mathrm{x}(R_{\mathsf{b}2}) = \mathrm{x}(\phi_{\mathsf{b}2}(P_{\mathsf{b}} - [2]Q_{\mathsf{b}})) \\
\mathrm{x}(P_{\mathsf{b}3}) = \mathrm{x}(\phi_{\mathsf{b}3}(P_{\mathsf{b}} + [3]Q_{\mathsf{b}})), & \mathrm{x}(Q_{\mathsf{b}3}) = \mathrm{x}(\phi_{\mathsf{b}3}([4]Q_{\mathsf{b}})), & \mathrm{x}(R_{\mathsf{b}3}) = \mathrm{x}(\phi_{\mathsf{b}3}(P_{\mathsf{b}} - Q_{\mathsf{b}})).
\end{array}$$

Notice that the above points to be evaluated imply six extra x-only point additions for computing $\mathrm{x}(P_{\mathsf{b}0})$, $\mathrm{x}(P_{\mathsf{b}1})$, $\mathrm{x}(P_{\mathsf{b}2})$, $\mathrm{x}(P_{\mathsf{b}3})$, $\mathrm{x}(R_{\mathsf{b}0})$, $\mathrm{x}(R_{\mathsf{b}1})$, $\mathrm{x}(R_{\mathsf{b}2})$, and $\mathrm{x}(R_{\mathsf{b}3})$. We formalize the running time of $\mathsf{Searcher}_4$ through Lemma 2.

Lemma 2. *Let $p = 4^a \cdot f - 1$ be a prime with $a \geq \log_4\left(\frac{\#\mathbb{S}_{p^2}}{\#\mathbb{S}_p}\right)$, and let $h = \log_4\left(\frac{\#\mathbb{S}_{p^2}}{\#\mathbb{S}_p}\right)$. The 4-isogeny variant of the algorithm in Figure 3 inspects at most $\frac{4}{3}\left(\frac{\#\mathbb{S}_{p^2}}{\#\mathbb{S}_p}\right)$ curves. In addition, the algorithm instantiated over the Kummer line has a time complexity bounded by*

$$82 \cdot \frac{\#\mathbb{S}_{p^2}}{\#\mathbb{S}_p}\ \mathsf{mul}.$$

Proof. Let us focus on the worst-case scenario (i.e., we inspect all the nodes in the 4-isogeny subtree). This time, inspecting the 4-isogeny subtree performs

- $\sum_{k=0}^{h} 4^k = \frac{1}{3} \cdot (4^{h+1} - 1) \approx \frac{4}{3} \cdot 4^h$ subfield checks,
- $\sum_{k=0}^{h-2} (6(h-k-1)+1) \cdot 4^k = \frac{2}{3} \cdot (4^h - 3h - 1) \approx \frac{2}{3} \cdot 4^h$ point doublings,
- $2 \cdot 4^{n-1} + \sum_{k=0}^{h-2} 8 \cdot 4^k = \frac{2}{3} \cdot 4^h + \frac{2}{3}(4^h - 4) \approx \frac{4}{3} \cdot 4^h$ point additions,
- $\sum_{k=0}^{h-2} 12 \cdot 4^k = 4^h - 4 \approx 3 \cdot 4^h$ isogeny evaluations, and
- $\sum_{k=0}^{h-1} 4 \cdot 4^k = \frac{4}{3} \cdot (4^h - 1) \approx \frac{4}{3} \cdot 4^h$ isogeny codomain calculations.

Then, similarly to Lemma 1, we obtain a running time bounded by

$$\mathsf{runtime}_4 = \frac{974}{12} \cdot 4^h\ \mathsf{mul} \approx 82 \cdot 4^d\ \mathsf{mul} = 82 \cdot \left(\frac{\#\mathbb{S}_{p^2}}{\#\mathbb{S}_p}\right) \mathsf{mul}. \tag{2}$$

However, differently from the algorithm $\mathsf{Searcher}_2$, observe that the algorithm $\mathsf{Searcher}_4$ inspects $\frac{1}{3} \cdot (4^{h+1} - 1)$ curves. If we restrict the search to the condition

$$h = \log_4\left(\frac{\#\mathbb{S}_{p^2}}{\#\mathbb{S}_p}\right) - 1$$

then we end up by inspecting only $\frac{1}{3} \cdot (4^h - 1)$ curves, which is less than $\frac{\#\mathbb{S}_{p^2}}{\#\mathbb{S}_p}$. Therefore, using

$$h = \log_4\left(\frac{\#\mathbb{S}_{p^2}}{\#\mathbb{S}_p}\right)$$

for $\mathsf{Searcher}_4$ implies inspecting $\frac{4}{3}\left(\frac{\#\mathbb{S}_{p^2}}{\#\mathbb{S}_p}\right)$ curves. □

Remark 1. We note that Lemma 2 does not include the inspection of the intermediate 2-isogeny nodes on the 4-isogeny tree. We discuss the impact of including this extra check in Appendix A.1.

4.2 Applicability to primes of the form $p = d^a \cdot f - 1$

It is worth noting that the algorithm $\mathsf{Searcher}_2$ can be easily extended to any d-isogeny, with the key difference that each node in the tree does not have two children but d children. For simplicity, we label the d-variant algorithm by $\mathsf{Searcher}_d$. If d is a small integer number, then $\mathsf{Searcher}_d$ has a cost of $\mathcal{O}(d^h)$ field multiplications.

Setup when employing d-isogenies. Following the notation from Fig. 3, the input must satisfy $\mathcal{E}[d^h] = \langle P, Q\rangle$. In addition, the integer n determines the length of the string $\mathsf{b} \in \{0, \ldots, d-1\}^n$, and we denote $\mathbf{z}_\mathsf{b} = \{\mathcal{E}, \mathsf{b}, \mathrm{x}(P_\mathsf{b}), \mathrm{x}(Q_\mathsf{b}), \mathrm{x}(R_\mathsf{b})\}$ where $R_\mathsf{b} = P_\mathsf{b} - Q_\mathsf{b}$.

Calculating Codomain d-isogenies when $n < (h-1)$. We need to calculate d different d-isogenies:

$$\phi_{\mathsf{b}j} \colon \mathcal{E}_\mathsf{b} \to \mathcal{E}_{\mathsf{b}j} \text{ with } \ker \phi_{\mathsf{b}j} = \langle [d^{h-n-1}](P_\mathsf{b} + [j]Q_\mathsf{b})\rangle \text{ for each } j \coloneqq 0, \ldots, d-1.$$

This time, the cost of computing the above d isogenies includes $3(h-n-1)$ x-only point multiplication by d and $(d-1)$ x-only point additions.

Calculating Codomain d-isogenies when $n = (h-1)$. We need to calculate d different d-isogenies:

$$\phi_{\mathsf{b}j} \colon \mathcal{E}_\mathsf{b} \to \mathcal{E}_{\mathsf{b}j} \text{ with } \ker \phi_{\mathsf{b}j} = \langle P_\mathsf{b} + [j]Q_\mathsf{b}\rangle \text{ for each } j \coloneqq 0, \ldots, d-1.$$

Similarly, the cost of computing the above d isogenies only includes $(d-2)$ x-only point additions.

Pushing Points Through d-isogenies. We need to push a total of $3d$ x-only points through the isogenies:

$$\begin{aligned} &\mathrm{x}(P_{\mathsf{b}j}) = \mathrm{x}(\phi_{\mathsf{b}j}(P_\mathsf{b} + [j]Q_\mathsf{b})), \qquad \mathrm{x}(Q_{\mathsf{b}j}) = \mathrm{x}(\phi_{\mathsf{b}j}([d]Q_\mathsf{b})), \text{ and} \\ &\mathrm{x}(R_{\mathsf{b}j}) = \mathrm{x}(\phi_{\mathsf{b}j}(P_\mathsf{b} - [d-j]Q_\mathsf{b})) \end{aligned}$$

for each $j \coloneqq 0, \ldots, d-1$. Those calculations includes $2(d-1)$ x-only point additions.

On the running time of $\mathsf{Searcher}_{12}$. To illustrate the impact of the algorithm $\mathsf{Searcher}_d$, let us focus on primes of the form $p = (4 \cdot 3)^a \cdot f - 1$, with $f \in \mathbb{N}$ and $a \geq \frac{1}{2}\log_{12} p$ (e.g., this kind of primes are used in [CSCD+23, DLRW24]). Hence, let us take $d = 12$ and consider the following. [7]

- A point multiplication by twelve requires calculating two-point doubling and one-point tripling. This gives a cost of $15\mathsf{M} + 9\mathsf{S}$ field operations.
- A 12-isogeny can be efficiently decomposed as the composition of a 4-isogeny followed by a 3-isogeny:
  - Computing a 12-isogeny curve codomain includes one point tripling, one 4-isogeny curve codomain calculation, one 4-isogeny evaluation, and one last 3-isogeny curve codomain calculation. This gives a cost of $15\mathsf{M} + 14\mathsf{S}$ field operations.

[7] We focus on 12-isogenies instead of 6-isogenies since we experimentally observed about the same complexities. In particular, we consider that for primes of the form $p = 2^a \cdot 3^b \cdot f - 1$ we have $a \approx b \log_2 3 \approx 1.58b$, and using $d = 12$ allows us to inspect a (slightly) bigger number of curves than when employing $d = 6$, at a negligible impact on the running time.

– Pushing a point through a 12-isogeny implies pushing the point through a 4-isogeny and a 3-isogeny. This gives a cost of $10\mathsf{M} + 4\mathsf{S}$ field operations.

We formalize the running time of $\mathsf{Searcher}_{12}$ through Lemma 3.

Lemma 3. *Let $p = 12^a \cdot f - 1$ be a prime with $a \geq \log_{12}\left(\frac{\#\mathbb{S}_{p^2}}{\#\mathbb{S}_p}\right)$, and let $h = \log_{12}\left(\frac{\#\mathbb{S}_{p^2}}{\#\mathbb{S}_p}\right)$. The 12-isogeny variant of the algorithm in Figure 3 inspects at most $\frac{12}{11}\left(\frac{\#\mathbb{S}_{p^2}}{\#\mathbb{S}_p}\right)$ curves. In addition, the algorithm instantiated over the Kummer line has a time complexity bounded by*

$$127 \cdot \frac{\#\mathbb{S}_{p^2}}{\#\mathbb{S}_p}\ \mathsf{mul}.$$

Proof. Let us focus on the worst-case scenario (i.e., we inspect all the nodes in the 12-isogeny subtree). This time, inspecting the 12-isogeny subtree performs

– $\sum_{k=0}^{h} 12^k = \frac{1}{11} \cdot (12^{h+1} - 1) \approx \frac{12}{11} \cdot 12^h$ subfield checks,

– $\sum_{k=0}^{h-2} 3(h-k-1) \cdot 12^k = \frac{3}{121} \cdot (12^h - 11h - 1) \approx \frac{3}{121} \cdot 12^h$ point multiplications by twelve,

– $10 \cdot 12^{h-1} + \sum_{k=0}^{h-2} 33 \cdot 12^k = \frac{10}{12} \cdot 12^h + \frac{1}{4}(12^h - 12) \approx \frac{13}{12} \cdot 12^h$ point additions,

– $\sum_{k=0}^{h-2} 36 \cdot 12^k = \frac{3}{11} \cdot (12^h - 12) \approx \frac{3}{11} \cdot 12^h$ isogeny evaluations, and

– $\sum_{k=0}^{h-1} 12 \cdot 12^k = \frac{12}{11} \cdot (12^h - 1) \approx \frac{12}{11} \cdot 12^h$ isogeny codomain calculations.

Then, running time of the algorithm $\mathsf{Searcher}_{12}$ is bounded by

$$\mathsf{runtime}_{12} = \frac{183460}{1452} \cdot 12^h\ \mathsf{mul} \approx 127 \cdot 12^h\ \mathsf{mul} = 127 \cdot \left(\frac{\#\mathbb{S}_{p^2}}{\#\mathbb{S}_p}\right)\ \mathsf{mul}. \tag{3}$$

However, differently from the algorithms $\mathsf{Searcher}_2$ and $\mathsf{Searcher}_4$, observe that the algorithm $\mathsf{Searcher}_{12}$ inspects $\frac{1}{11} \cdot (12^{h+1} - 1)$ curves, which implies inspecting $\frac{12}{11}\left(\frac{\#\mathbb{S}_{p^2}}{\#\mathbb{S}_p}\right)$ curves. □

4.3 Adapting the Fast Subfield Root Detection in $\mathcal{X}(\overline{\mathbb{F}}_p, \ell)$

Our description of algorithm $\mathsf{Searcher}_d$ explicitly works over the elliptic curves, which would require the calculation of $j(\mathcal{E}_\mathsf{b})$ (adding a field inversion calculation) if using algorithm $\mathsf{NeighbourInFp}$. For instance, from the optimized instantiation of algorithm $\mathsf{Searcher}_d$ over the Kummer line (see Section 4), we can calculate the numerator u and denominator v of each j-invariant $j = u \cdot v^{-1}$. Then, we cannot directly employ algorithm $\mathsf{NeighbourInFp}$ from [SCS22] without the calculation of v^{-1}. [8] Including that field inversion would increase the overall complexity of our searcher algorithm by a logarithm factor $\log_2 p$, therefore increasing the runtime to $\tilde{\mathcal{O}}(p^{1/2})$.

However, to avoid the calculation of the inverse of v, we next adapt the procedure $\mathsf{NeighbourInFp}$ to our case study. Let us assume we have $u, v \in \mathbb{F}_{p^2}$ such that $v \neq 0$ and $j(\mathcal{E}) = \frac{u}{v}$ is the j-invariant of a supersingular elliptic curve $\mathcal{E}$, and consider the polynomial

$$\Delta_\ell(x, u, v) = v^{d_\ell} \cdot \Phi_\ell\left(x, \frac{u}{v}\right).$$

[8] Corte-Real Santos, Costello, and Shi highlight that having the j-invariants explicitly computed is critical in their subfield root detection algorithm [SCS22, Remark 2].

Notice the coefficients of $\Delta_\ell(x,u,v)$ do not require the explicit value of v^{-1}, and we have

$$g_1 = \frac{1}{2}\left(\Phi_\ell + \pi(\Phi_\ell)\right) = \frac{1}{2w}\left(\pi(v^{d_\ell})\cdot\Delta_\ell + v^{d_\ell}\cdot\pi(\Delta_\ell)\right), \text{ and}$$
$$g_2 = -\frac{i}{2}\left(\Phi_\ell - \pi(\Phi_\ell)\right) = -\frac{i}{2w}\left(\pi(v^{d_\ell})\cdot\Delta_\ell - v^{d_\ell}\cdot\pi(\Delta_\ell)\right),$$

where $w = (v\cdot\pi(v))^{d_\ell}$ and $\pi\colon x \mapsto x^p$ denotes the Frobenius map. Re-writing g_1 and g_2 in terms of $\Theta_\ell = \pi(v)^{d_\ell}\cdot\Delta_\ell$ gives

$$g_1 = \frac{1}{2w}\left(\Theta_\ell + \pi(\Theta_\ell)\right) \quad \text{and} \quad g_2 = -\frac{i}{2w}\left(\Theta_\ell - \pi(\Theta_\ell)\right).$$

Since $\gcd(g_1,g_2) = \gcd(w\cdot g_1, w\cdot g_2)$ holds, we can evaluate Θ_ℓ at (u,v) as follows:

- Calculate $u^2,\ldots,u^{d_\ell}, v^2,\ldots,v^{d_\ell}$, and $t = \pi(v)^{d_\ell} = \pi(v^{d_\ell})$.
- Compute $z_m = t\cdot u^n\cdot v^m$ for each $n,m \in \{0\ldots d_\ell\}$ such that $d_\ell = n+m$.
- Multiply each coefficient z_m by the corresponding coefficients of Φ_ℓ.

The above third step essentially costs what EvaluateModularPolynomial does for calculating $\Phi_\ell(x,j)$ but this time multiplying by z_m instead of j^m for obtaining $\Theta_\ell(x,u,v)$. Then, we can evaluate Θ_ℓ at (u,v) by performing $(9d_\ell(d_\ell-1)+9(d_\ell-1))$mul operations. On the other hand, InverseFreeGCD$(w\cdot g_1, w\cdot g_2)$ still takes $(2d_\ell+1)(d_\ell+1)-6$ multiplications in $\mathbb{F}_p$. [9] Thus, our modified subfield root detection algorithm ScaledNeighbourInFp(ℓ,j) requires at most $(11d_\ell^2+3d_\ell-14)$mul operations. We give a high-level description of the algorithm ScaledNeighbourInFp in Figure 4.

ScaledNeighbourInFp(ℓ,u,v)

1 : $f^\star \leftarrow$ EvaluateScaledModularPolynomial(ℓ,u,v)
2 : Compute $g_1^\star = \frac{1}{2}(f^\star + \pi(f^\star))$
3 : Compute $g_2^\star = -\frac{i}{2}(f^\star - \pi(f^\star))$
4 : **return** InverseFreeGCD$(g_1^\star, g_2^\star)$

Figure 4: Description of our modified subfield root detection algorithm. The input must satisfy that $j(\mathcal{E}) = \frac{u}{v}$ for some supersingular curve $\mathcal{E}$ defined over $\mathbb{F}_{p^2}$. The subroutine EvaluateScaledModularPolynomial(ℓ,j) evaluates $\Theta_\ell(x,y)$ at (u,v) as described above.

4.3.1 Optimized Subfield Curve Search Procedure.

Given a list of t small odd primes $L = [\ell_1,\ell_2,\ldots,\ell_t]$, we suggest modifying the DFS algorithm in Figure 3 by including the subfield curve inspection for each $\ell \in L$ in Line 4. In that way, we increase the number of elliptic curves inspected, i.e., it allows us to detect if there is one ℓ-isogenous subfield curve among the $(\ell+1)$ possibilities. That optimization helps to increase the number of inspected curves in Searcher$_d$ by a factor of $(1+\sum_{\ell\in L} d_\ell)$ times bigger. Therefore, we can decrease the value of h to (at most)

$$h = \log_d\left(\frac{\#\mathbb{S}_{p^2}}{\#\mathbb{S}_p}\right) - \left\lfloor \log_d\left(1+\sum_{\ell\in L} d_\ell\right)\right\rceil$$

and still inspect around $\frac{\#\mathbb{S}_{p^2}}{\#\mathbb{S}_p}$ curves.

[9] This is because $\Theta_\ell(x,u,v) = w\cdot\Phi_\ell(x,\frac{u}{v})$ with $w\in\mathbb{F}_p$ hold, and its leading coefficient is w itself.

4.3.2 Optimizing the choice of each ℓ_k

The situation here differs from [SCS22] because we do not need to calculate square roots to compute 2-isogenous curves. However, we can isolate the cost associated with inspecting through d-isogenies in algorithm $\mathsf{Searcher}_d$, then split the total associated cost to ScaledNeighbourInFp, and maximize the number of inspected curves as much as possible.

In what follows, we label the modified algorithm, which includes the subfield curve search, by $\mathsf{SuperSearcher}_d$ to distinguish it from the baseline algorithm $\mathsf{Searcher}_d$. We adopt the subfield cost for each algorithm step as in [SCS22] to better understand the difference in cost of inspecting $\mathcal{X}(\overline{\mathbb{F}}_p, \ell)$ when employing NeighbourInFp and ScaledNeighbourInFp; more precisely, we take that cost as

$$\mathsf{cost}^*_\ell = \frac{1}{d_\ell}(\text{cost of ScaledNeighbourInFp}) \leq \frac{1}{d_\ell}(11d_\ell^2 + 3d_\ell - 14)\ \mathsf{mul}.$$

Table 3 presents and compares the concrete cost for inspecting $\mathcal{X}(\overline{\mathbb{F}}_p, \ell)$ when employing NeighbourInFp and ScaledNeighbourInFp. Observe that the procedure ScaledNeighbourInFp is slightly costlier than NeighbourInFp, but we highlight that the algorithm $\mathsf{Searcher}_d$ with NeighbourInFp adds logarithmic factors in the overall running time while with ScaledNeighbourInFp does not. We thus consider $(\mathsf{c}, d) \in \{(132, 2), (82, 4), (127, 12)\}$ and the running time of algorithm $\mathsf{SuperSearcher}_d$ to be bounded by

$$\mathsf{runtime}^*_{\mathsf{d}} = \underbrace{\frac{1}{d^{\left\lfloor \log_d\left(1+\sum_{\ell\in L} d_\ell\right)\right\rfloor}}\left(\mathsf{c} + \frac{d}{d-1}\cdot\sum_{\ell\in L}\overbrace{d_\ell \cdot \mathsf{cost}^*_\ell}^{\text{ScaledNeighbourInFp}}\right)}_{\gamma(L)} \cdot d^h\ \mathsf{mul}, \tag{4}$$

Table 3: The row corresponding to cost_ℓ presents the values from [SCS22, Table 3].

ℓ:	3	5	7	11	13	17	19	23	29	31	37	41	43
d_ℓ:	4	6	8	12	14	18	20	24	30	32	38	42	44
cost_ℓ:	16.3	24.5	32.6	48.8	56.8	72.8	80.9	96.9	120.9	128.9	152.9	168.9	176.9
$\mathsf{cost}^\star_\ell$:	25.8	35.8	43.9	61.4	69.2	85.9	93.8	110.2	134.4	142.2	166.3	182.5	190.4

Observe that $\gamma(L)$ does not depend on neither $\log_2 p$ nor d; therefore, we can consider it a constant factor. With that in mind, we want to minimize the value of $\gamma(L) \leq \mathsf{c}$. We combine the concrete costs from Table 3 to select the best candidate of L with at most thirteen elements (any other larger prime number ℓ implies a higher number of field operations when inspecting $\mathcal{X}(\overline{\mathbb{F}}_p, \ell)$). More precisely, we perform a brute-force search on $\mathbf{b} \in \{0, 1\}^{13}$ to determine

$$\operatorname*{argmin}_{\mathbf{b}}\ \gamma(L_{\mathbf{b}}),$$

where $L_{\mathbf{b}}$ denotes the list having the ith prime ℓ_i if the ith bit of $\mathbf{b}$ is nonzero.

Remark 2. Given that algorithm $\mathsf{SuperSearcher}_d$ inspects

$$N = \frac{d}{d-1}\cdot d^h \cdot (1 + \sum_{\ell\in L} d_\ell)$$

curves, where $h = \log_d\left(\frac{\#\mathbb{S}_{p^2}}{\#\mathbb{S}_p}\right) - \lfloor \log_d(1 + \sum_{\ell\in L} d_\ell)\rceil$, we consider that if N is smaller than $\frac{\#\mathbb{S}_{p^2}}{\#\mathbb{S}_p}$, then algorithm $\mathsf{SuperSearcher}_d$ needs to increase the value of h by one to inspect at least $\frac{\#\mathbb{S}_{p^2}}{\#\mathbb{S}_p}$ curves; in other words: if $N < \frac{\#\mathbb{S}_{p^2}}{\#\mathbb{S}_p}$ then the value of γ needs to be multiplied by d.

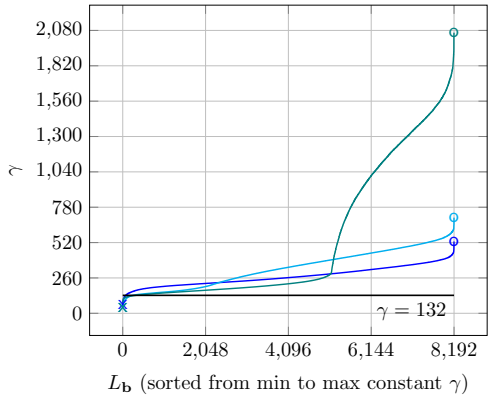


Figure 5: Value of $\gamma(L_{\mathbf{b}})$ for each $\mathbf{b} \in \{0,1\}^{13}$. The x-axis represents the choice of the list $L_{\mathbf{b}}$ sorted from the one with minimal constant $\min \gamma(L_{\mathbf{b}})$ to the one with maximal constant $\max \gamma(L_{\mathbf{b}})$. The y-axis determines the constant $\gamma(L_{\mathbf{b}})$ as given in Equation (4). We mark the maximum $\max \gamma(L_{\mathbf{b}})$ by o, and the minimum $\min \gamma(L_{\mathbf{b}})$ with x. The curve in blue ink color corresponds with algorithm SuperSearcher, while the one with cyan (resp. teal) ink presents the values for algorithm $\mathsf{SuperSearcher}_4$ (resp. $\mathsf{SuperSearcher}_{12}$). The big jumps on the graph concerning $\mathsf{SuperSearcher}_{12}$ come from the values of γ being multiplied by $d = 12$ as stated in Remark 2.

We plot all the values of $\gamma(L_{\mathbf{b}})$ for each $\mathbf{b} \in \{0,1\}^{13}$ in Figure 5. In particular, we obtain that the best choice for L is

- $L_{10} = [3,7]$ with $\gamma(L_{10}) = 65$ when performing the algorithm $\mathsf{SuperSearcher}_2$,
- $L_{10} = [3,7]$ with $\gamma(L_{10}) = 42.96$ when using the algorithm $\mathsf{SuperSearcher}_4$,
- $L_6 = [3,5]$ with $\gamma(L_6) = 39.49$ when employing the algorithm $\mathsf{SuperSearcher}_{12}$.

Additionally, for each $(d, L) \in \{(2, L_{10}), (4, L_{10}), (12, L_6)\}$, we have that the number of curves inspected in $\mathsf{SuperSearcher}_d$ is bounded by

$$\frac{d}{d-1} \cdot \left(\frac{1 + \sum_{\ell \in L} d_\ell}{d^{\lfloor \log_d (1 + \sum_{\ell \in L} d_\ell) \rceil}} \right) \cdot \left(\frac{\#\mathbb{S}_{p^2}}{\#\mathbb{S}_p} \right) \geq \frac{\#\mathbb{S}_{p^2}}{\#\mathbb{S}_p}. \tag{5}$$

Remark 3. We emphasize that our results should be taken as asymptotic improvements: in the matter that the bitlength of p increases, the bigger improvements for the subfield curve searching. To better illustrate, Table 4 presents the number of field multiplications over $\mathbb{F}_p$ per inspected curves for both algorithms SuperSolver and SuperSearcher. For simplicity, we take the best choice of L that maximizes the number of inspected curves in SuperSolver, and compare it with $\mathsf{SuperSearcher}_4$.

4.4 On the Success Probability and Time Complexity of the Algorithm in Practice

We stress that for each $d \in \{2, 4, 12\}$ the algorithms $\mathsf{Searcher}_d$ and $\mathsf{SuperSearcher}_d$ have two possible outcomes: If the algorithm reaches a subfield curve, then it outputs a valid isogeny path. Otherwise, it returns $\perp$. However, if we consider that each inspected curve through algorithm A has a probability $\frac{\#\mathbb{S}_p}{\#\mathbb{S}_{p^2}}$ of being a subfield curve, then we can estimate the success probability of algorithm A outputting a valid isogeny path with endpoint a subfield curve. We estimate this probability in Theorem 1.

Table 4: Number of field multiplications over $\mathbb{F}_p$ per inspected curves for primes of the form $p = 2^a \cdot f - 1$. For SuperSolver algorithm, we run the provided code of [SCS22] and take the best choice of L that maximize the number of inspected curves. The columns concerning "This work" correspond to algorithm $\mathsf{SuperSearcher}_4$.

a	f	[SCS22]		This work		
		L	Mult. per visited curve	L	Mult. per visited curve	Improv. Factor
50	11	$[3, 5]$	35.42	$[3, 7]$	39.73	0.89x
100	233	$[3, 5, 7]$	46.03	$[3, 7]$	39.74	1.16x
150	17	$[3, 5, 7, 9, 11]$	51.78	$[3, 7]$	39.76	1.30x
200	449	$[3, 5, 7, 9, 11]$	56.60	$[3, 7]$	39.78	1.42x
250	1091	$[3, 5, 7, 9, 11, 13]$	60.16	$[3, 7]$	39.81	1.51x
300	719	$[3, 5, 7, 9, 11, 13]$	63.68	$[3, 7]$	39.85	1.60x
350	557	$[3, 5, 7, 9, 11, 13]$	67.06	$[3, 7]$	39.89	1.68x
400	2339	$[3, 5, 7, 9, 11, 13]$	70.58	$[3, 7]$	39.94	1.77x
450	3083	$[3, 5, 7, 9, 11, 13, 17]$	73.74	$[3, 7]$	39.99	1.84x
500	197	$[3, 5, 7, 9, 11, 13, 17]$	76.24	$[3, 7]$	40.05	1.90x
550	83	$[3, 5, 7, 9, 11, 13, 17]$	78.85	$[3, 7]$	40.11	1.96x
600	293	$[3, 5, 7, 9, 11, 13, 17, 19]$	81.39	$[3, 7]$	40.19	2.03x
650	113	$[3, 5, 7, 9, 11, 13, 17, 19]$	83.43	$[3, 7]$	40.27	2.07x
700	3533	$[3, 5, 7, 9, 11, 13, 17, 19]$	85.70	$[3, 7]$	40.36	2.12x
750	1151	$[3, 5, 7, 9, 11, 13, 17, 19]$	87.76	$[3, 7]$	40.47	2.17x
800	347	$[3, 5, 7, 9, 11, 13, 17, 19]$	89.76	$[3, 7]$	40.59	2.21x

Theorem 1. *Let $\#\mathbb{S}_{p^2}$ be the number of supersingular elliptic curves defined over $\mathbb{F}_{p^2}$ up to isomorphism, and $\#\mathbb{S}_p$ be the number of those defined over $\mathbb{F}_p$. Consider a small constant $c \geq 1$, $d \in \{2, 4, 12\}$ and*

$$\rho = \frac{\#\mathbb{S}_p}{\#\mathbb{S}_{p^2}} \in (0, 1).$$

Assume that each curve inspected in the algorithm $\mathsf{A} \in \{\mathsf{Searcher}_d, \mathsf{SuperSearcher}_d\}$ is a subfield curve with probability ρ. Then, after $t = c\frac{\#\mathbb{S}_{p^2}}{\#\mathbb{S}_p}$ different curves inspected in the algorithm A, the probability of encountering at least one subfield curve is greater than or equal to $1 - e^{-c}$.

Proof. Let X_i be the indicator random variable for the event that the ith inspected curve is a subfield curve, that is

$$X_i = \begin{cases} 1 & \text{the } i\text{-th inspected curve through algorithm } \mathsf{A} \text{ is a subfield curve} \\ 0 & \text{otherwise} \end{cases}$$

By assumption, we have $\mathsf{Pr}[X_i = 1] = \rho$ and $\mathsf{Pr}[X_i = 0] = 1 - \rho$. Let $X = \sum_{i=1}^{t} X_i$. Since each trial succeeds independently with probability ρ, we find that

$$\mathsf{Pr}\left[\begin{array}{l}\text{no subfield curve is} \\ \text{reached after } t \text{ steps}\end{array}\right] = \mathsf{Pr}[X = 0] = (1 - \rho)^t.$$

Hence, the probability of reaching at least one subfield curve after $t = c\frac{\#\mathbb{S}_{p^2}}{\#\mathbb{S}_p} = \frac{c}{\rho}$ steps is

$$\mathsf{Pr}\left[\begin{array}{l}\text{at least one subfield curve} \\ \text{is reached after } t \text{ steps}\end{array}\right] = \mathsf{Pr}[X \geq 1] = 1 - (1 - \rho)^t = 1 - (1 - \rho)^{c/\rho}.$$

Using the inequality $1 - x \leq e^{-x}$ for $x \in (0, 1)$, we verify that $(1 - \rho)^{c/\rho} \leq e^{-c}$ since $c \geq 1$, and therefore

$$\mathsf{Pr}\left[\begin{array}{l}\text{at least one subfield curve} \\ \text{is reached after } t \text{ steps}\end{array}\right] \geq 1 - e^{-c}.$$

□

On the Success Probability and Average Time Complexity: At this point, we have all the ingredients to estimate the probability that each algorithm $\mathsf{A} \in \{\mathsf{Searcher}_d, \mathsf{SuperSearcher}_d\}_{d\in\{2,4,12\}}$ reaches a subfield curve. On the one hand, for algorithm $\mathsf{Searcher}_d$ we see that [10]

$$c = \begin{cases} 1 & \text{if } d = 2, \\ \frac{4}{3} & \text{if } d = 4, \\ \frac{12}{11} & \text{if } d = 12. \end{cases} \quad \Rightarrow \quad \zeta \geq \begin{cases} 1 - e^{-1} \gtrsim 0.63 & \text{if } d = 2, \\ 1 - e^{-\frac{4}{3}} \gtrsim 0.73 & \text{if } d = 4, \\ 1 - e^{-\frac{12}{11}} \gtrsim 0.66 & \text{if } d = 12. \end{cases}$$

Conversely, for algorithm $\mathsf{SuperSearcher}_d$ we find that [11]

$$c = \begin{cases} \frac{13}{8} & \text{if } d = 2, \\ \frac{13}{12} & \text{if } d = 4, \\ 1 & \text{if } d = 12. \end{cases} \quad \Rightarrow \quad \zeta \geq \begin{cases} 1 - e^{-\frac{13}{8}} \gtrsim 0.80 & \text{if } d = 2, \\ 1 - e^{-\frac{13}{12}} \gtrsim 0.66 & \text{if } d = 4, \\ 1 - e^{-1} \gtrsim 0.63 & \text{if } d = 12. \end{cases}$$

As a main consequence of Theorem 1, we see that algorithm A reaches a subfield curve with probability greater than or equal to $\zeta = 1 - e^{-c}$ for some constant $c \geq 1$. If algorithm A returns $\perp$ for a given instance $\mathbb{I}$. Then, we can randomize the given instance $\mathbb{I}$ by translating it through a random isogeny walk to get another instance $\mathbb{J}$ as input for the algorithm A. On average, we expect to repeat this procedure $\frac{1}{\zeta}$ times in order to obtain the first success in finding a subfield curve. On that basis, consider the number $N = \frac{1}{\zeta}$ of trials and that

- the first $N - 1$ trials concern calls to A that return $\perp$, and
- the last trial determines the call to A that encounters a subfield curve.

For the first $N - 1$ calls to A, we inspect the maximum number of curves since they return $\perp$. For the last call to A, which succeeds in finding a subfield curve, we expect that algorithm A inspects (on average) half of the maximum inspected curves to reach a subfield curve since in the best-case scenario the input was a subfield curve and in the worst-case scenario the last curve is a subfield curve. Let t_f be the number of inspected curves for each of the first $N - 1$ trials, and t_s the average number of inspected curves for the last trial. Then

$$E\left[\begin{array}{l}\text{inspected curves}\\ \text{until reach a}\\ \text{subfield curve}\end{array}\right] = E[(N-1)t_f + t_s] = (N-1)E[t_f] + E[t_s] = c\left(\frac{1}{\zeta} - \frac{1}{2}\right)\frac{\#\mathbb{S}_{p^2}}{\#\mathbb{S}_p},$$

where $c \in [1, 2)$. On the other hand, let r_f be the running time of algorithm A for each of the first $N - 1$ trials, and r_s the average running time for the last trial. Then

$$E\left[\begin{array}{l}\text{running time}\\ \text{until reach a}\\ \text{subfield curve}\end{array}\right] = E[(N-1)r_f + r_s] = (N-1)E[r_f] + E[r_s] = \gamma\left(\frac{1}{\zeta} - \frac{1}{2}\right)\frac{\#\mathbb{S}_{p^2}}{\#\mathbb{S}_p},$$

where γ is the constant in the complexity as detailed in previous section for each algorithm.

[10] The values of c for $\mathsf{Searcher}$ are taken from Lemmas 1 to 3.
[11] The values of c for $\mathsf{SuperSearcher}$ are obtained through Equation (5).

5 Experimental Results

Since algorithms $\mathsf{Searcher}_d$ and $\mathsf{SuperSearcher}_d$ for $d \in \{2, 4\}$ perform on top of 2-isogenies (and 3-isogenies) between Montgomery curves, we generate "random" curve instances by walking through the 5-isogeny graph to avoid bias. We take $j_0 = 287496$ as the supersingular j-invariant origin and calculate a random 5-isogeny walk $j_0 \xrightarrow{\phi_1} j_1 \xrightarrow{\phi_2} \cdots \xrightarrow{\phi_n} j_n$ of length $n = \log_5 p$. Next, we compute a Montgomery curve $\mathcal{E}\colon y^2 = x^3 + Ax^2 + x$ with j-invariant j_d and find a basis $\{P, Q\}$ of the torsion subgroup $\mathcal{E}[d^h]$ with $d \in \{2, 4, 12\}$. Given that $\#\mathbb{S}_p$ is significantly harder to estimate theoretically, without a specific p in mind, we take $h = \frac{1}{2}\log_2 p - \epsilon$ for some specific values of ϵ. Our experiments confirm our theoretical results, and highlight the improvements against [SCS22]. In particular, our experiments illustrate a greater success probability than the theoretical lower bound (most of the time we reach the upper bound probability $\zeta = 1$), which results in a lesser running time for our algorithm. We provide a proof-of-concept implementation using SageMath [The25], available at [Com26].

Experiments concerning $\mathsf{Searcher}_d$ and $\mathsf{SuperSearcher}_d$: We center our experiments on r-bits prime numbers of the form $p = 2^a \cdot f - 1$ with $a \approx r$ and $a \approx \frac{1}{2}r$. In particular, our experiments limit to $r \in [31, 50] \cap \mathbb{Z}$. We brute force search the interval $[31, 50] \cap \mathbb{Z}$ to get primes with $a \approx r$, and employ the extended Euclidean algorithm technique from [Cos20] for primes with $a \approx \frac{1}{2}r$. We include experiments for the algorithms $\mathsf{Searcher}_2$ and $\mathsf{SuperSearcher}_2$ for both cases $a \approx \frac{1}{2}r$ and $a \approx r$ (see Tables 5 and 6). Given that $\mathsf{Searcher}_4$ and $\mathsf{SuperSearcher}_4$ implicitly ask for $4 \mid a$, we limit our experiments to $a \approx r$ for these 4-variant algorithms (see Table 7). Additionally,

- For $\mathsf{Searcher}_2$, we take as inputs $\mathcal{E}$, $\mathrm{x}(P)$, $\mathrm{x}(Q)$ $\mathrm{x}(P - Q)$, and $h = \lfloor \frac{1}{2}\log_2 p \rfloor - 1$. For $\mathsf{Searcher}_4$, we use $h = \lfloor \frac{1}{2}\log_4 p \rfloor$.
- For $\mathsf{SuperSearcher}_2$, we take as inputs $\mathcal{E}$, $\mathrm{x}([2^4]P)$, $\mathrm{x}([2^4]Q)$ $\mathrm{x}([2^4](P - Q))$, and $h = \lfloor \frac{1}{2}\log_2 p \rfloor - 4$. For $\mathsf{SuperSearcher}_4$, we use $h = \lfloor \frac{1}{2}\log_4 p \rfloor - 2$.

Table 5: All the numbers correspond to the average of solving 100 random instances by using $\mathsf{Searcher}_2$ and $\mathsf{SuperSearcher}_2$. The Avg. γ values present the measured average value of γ; i.e., the measured average running time of each algorithm is $\gamma \cdot 2^n$ with $n = \frac{1}{2}\log_2 p$. The Meas. ζ values determine the success ratio of each experiment; i.e., the measured successful probability of algorithm $\mathsf{SuperSearcher}_2$.

Parameters									
$\log_2 p$:	32	35	37	38	40	41	42	43	50
a:	17	21	19	19	22	21	22	22	26
f:	19417	10645	140995	337825	152945	601135	861533	1924415	9365135
$\mathsf{Searcher}_2$									
Avg. γ:	20.2	25.7	30.8	21.4	14.0	21.7	22.2	24.3	15.8
Meas. ζ:	1.0	1.0	0.8	1.0	1.0	0.9	1.0	0.8	1.0
$\mathsf{SuperSearcher}_2$									
Avg. γ:	20.5	18.4	21.4	19.0	8.5	17.9	18.7	23.2	13.2
Meas. ζ:	0.9	1.0	1.0	0.9	1.0	1.0	1.0	0.9	1.0

Experiments concerning $\mathsf{Searcher}_{12}$ and $\mathsf{SuperSearcher}_{12}$: We include experiments for primes of the form $p = 2^a \cdot 3^b \cdot f - 1$ (such primes are used, for example, in [DLRW24]); we assume $f \in \{5, 7, 11\}$. We also perform experiments for $p = 12^a \cdot f - 1$ with $a \gtrapprox \frac{1}{2}\log_{12} p$ and f being a prime number (see Table 8). We brute-force search the interval $[29, 50] \cap \mathbb{Z}$ for finding these kinds of primes. In addition,

- For $\mathsf{Searcher}_{12}$, we take as inputs $\mathcal{E}$, $\mathrm{x}(P)$, $\mathrm{x}(Q)$ $\mathrm{x}(P - Q)$, and $h = \lfloor \frac{1}{2}\log_{12} p \rfloor$.

Table 6: The description is the same as in Table 5.

Parameters												
$\log_2 p$:	31	35	36	37	38	40	41	42	45	46	47	48
a:	31	32	34	31	33	35	36	37	39	41	40	42
f:	1	5	3	43	31	31	17	31	61	19	107	53
$\mathsf{Searcher}_2$												
Avg. γ:	21.4	23.4	9.3	40.3	24.8	31.6	29.9	21.5	40.6	16.9	25.1	29.2
Meas. ζ:	1.0	0.9	1.0	0.7	0.9	0.8	0.8	0.9	0.8	1.0	1.0	0.7
$\mathsf{SuperSearcher}_2$												
Avg. γ:	18.6	25.5	9.8	31.4	15.9	22.0	14.5	20.3	26.7	17.6	16.2	18.0
Meas. ζ:	0.9	0.8	1.0	0.9	1.0	1.0	1.0	1.0	0.8	1.0	1.0	1.0

Table 7: The description is the same as in Table 5, with the difference that these experiments correspond to algorithms $\mathsf{Searcher}_4$ and $\mathsf{SuperSearcher}_4$.

Parameters												
$\log_2 p$:	31	35	36	37	38	40	41	42	45	46	47	48
a:	31	32	34	31	33	35	36	37	39	41	40	42
f:	1	5	3	43	31	31	17	31	61	19	107	53
$\mathsf{Searcher}_4$												
Avg. γ:	14.4	16.0	5.0	37.4	18.5	28.2	19.0	8.7	33.1	7.8	16.6	16.9
Meas. ζ:	1.0	1.0	1.0	0.9	0.9	0.9	1.0	1.0	0.9	1.0	1.0	1.0
$\mathsf{SuperSearcher}_4$												
Avg. γ:	11.0	11.9	5.2	18.2	8.7	15.7	10.4	6.9	20.8	10.0	9.4	11.6
Meas. ζ:	1.0	1.0	1.0	0.9	0.9	1.0	1.0	0.9	0.8	0.9	1.0	1.0

- For $\mathsf{SuperSearcher}_{12}$, we take as inputs $\mathcal{E}$, x([12]P), x([12]Q) x([12]($P-Q$)), and $h = \lfloor \frac{1}{2} \log_{12} p \rfloor - 1$.

We also include experiments for the algorithms $\mathsf{Searcher}_2$ and $\mathsf{SuperSearcher}_2$ for these primes (see Table 9), and provide some additional graphs in Appendix A.2.

Table 8: The description is the same as in Table 5, with the difference that these experiments correspond to algorithms $\mathsf{Searcher}_{12}$ and $\mathsf{SuperSearcher}_{12}$.

Parameters								
$\log_2 p$:	30	34	36	40	42	44	48	50
a:	4	5	5	6	6	6	7	7
f:	32783	65599	262151	262231	1048793	4194493	4194329	16777499
$\mathsf{Searcher}_{12}$								
Avg. γ:	18.9	28.0	12.6	17.8	12.5	18.7	12.7	10.1
Meas. ζ:	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
$\mathsf{SuperSearcher}_{12}$								
Avg. γ:	4.9	7.4	6.5	11.3	7.6	6.9	6.5	4.0
Meas. ζ:	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0

Table 9: The description is the same as in Table 5.

Parameters							
$\log_2 p$:	29	34	36	37	38	39	45
a:	15	18	19	19	19	20	24
f:	15309	45927	98415	216513	413343	413343	1240029
$\mathsf{Searcher}_2$							
Avg. γ:	16.3	15.1	10.7	22.5	9.2	21.3	28.0
Meas. ζ:	1.0	1.0	1.0	1.0	1.0	1.0	0.9
$\mathsf{SuperSearcher}_2$							
Avg. γ:	10.9	13.4	11.5	15.0	14.3	27.0	18.1
Meas. ζ:	1.0	1.0	1.0	1.0	1.0	0.8	1.0

6 Concluding Remarks

Our results in Section 4, backed up with our experimental results from Section 5, highlight a time complexity of $\mathcal{O}(\frac{\#\mathbb{S}_{p^2}}{\#\mathbb{S}_p}) = \mathcal{O}(p^{1/2})$ field multiplications and memory complexity polynomial in $\log_2 p$ for both algorithms $\mathsf{Searcher}_d$ and $\mathsf{SuperSearcher}_d$. [12]

We emphasize that our results should be taken as cautionary advice for proposing conservative parameter sets in isogeny-based constructions with 2λ-bits primes of the form $p = 2^a \cdot f - 1$ with $a \geq \lambda$ and $\lambda \in \{128, 192, 256\}$. Our results suggest a smaller complexity compared to the state of the art. We present in Table 10 some concrete numbers to illustrate the impact of our results from Section 4. That table provides the expected average complexities for specific isogeny-based constructions.

Table 10: All numbers are given in $\log_2$ base and correspond to the expected average time complexity. We present complexities in terms of field multiplications; that is, they determine $\gamma \cdot p^{1/2}$ for some constant γ. For SuperSolver algorithm, we use the quantities i) $\gamma = 58$ for $\log_2 p \approx 256$, ii) $\gamma = 63.1$ for $\log_2 p \approx 384$, and iii) $\gamma = 69.1$ for $\log_2 p \approx 512$; These values of γ are taken from [SCS22, Table 7]. While for this work we consider $\mathsf{SuperSearcher}_4$ and take γ given by the success probability $\zeta \in \{0.66, 0.8, 0.9, 1.0\}$ according to Sections 4.4 and 5.

Scheme	$\log_2 p$	**[SCS22]**	**This work**			
			$\zeta = 0.66$	$\zeta = 0.8$	$\zeta = 0.9$	$\zeta = 1.0$
SQIsign v1.0 [CSCD+23]	254.0	132.9	132.4	132.0	131.7	131.4
	382.0	197.0	196.4	196.0	195.7	195.4
	510.0	261.1	260.4	260.0	259.7	259.4
SQIsignHD [DLRW24]	254.0	132.9	132.4	132.0	131.7	131.4
	389.0	200.5	199.9	199.5	199.2	198.9
	519.0	265.6	264.9	264.5	264.2	263.9
SQIsign2D-West [BDD+24]	251.0	131.4	130.9	130.5	130.2	129.9
	383.0	197.5	196.9	196.5	196.2	195.9
	505.0	258.6	257.9	257.5	257.2	256.9
SQIPrime [DF24]	254.0	132.9	132.4	132.0	131.7	131.4
	413.0	212.5	211.9	211.5	211.2	210.9
	508.0	260.1	259.4	259.0	258.7	258.4
SQIsign2D-East [NOC+24]	258.0	134.9	134.4	134.0	133.7	133.4
	386.0	199.0	198.4	198.0	197.7	197.4
	515.0	263.6	262.9	262.5	262.2	261.9
ApresSQI [SEMR24]	254.0	132.9	132.4	132.0	131.7	131.4
SQIsign v2.0 [AAA+25]	251.0	131.4	130.9	130.5	130.2	129.9
	383.0	197.5	196.9	196.5	196.2	195.9
	505.0	258.6	257.9	257.5	257.2	256.9

Since attacks are constantly improving (and our results are a clear example of this), potential advances in the cryptanalysis of isogeny-based constructions should be taken into account when selecting parameters, e.g., by including some margin in the bit-security estimates of future isogeny-based constructions. For instance, most of the competitors in the NIST call for additional post-quantum signature schemes align with the NIST security considerations of [Nat22, Section 4.B.3], and consider:

- NIST Security Level 1 asks for a classical complexity of 2^{143} bit operations.
- NIST Security Level 3 asks for a classical complexity of 2^{207} bit operations.
- NIST Security Level 5 asks for a classical complexity of 2^{272} bit operations.

Against these targets, as Table 10 shows, the security of the proposed parameter sets hinges entirely on the cost assigned to field arithmetic: under conservative (low) cost

[12] Our algorithms are highly parallelizable and extend to other primes (see Appendices A.3 and A.4).

estimates for a field multiplication, the estimated attack complexities fall slightly below the NIST security levels above, whereas more pessimistic estimates leave the parameters within their claimed categories.

Acknowledgements

We thank each anonymous reviewer for their valuable comments that helped to improve an early version of this manuscript. Additionally, we thank Loïc Bidoux, Mukul Kulkarni, Ricardo Pontaza-Rodas, and the SQIsign team for their insightful comments and discussions on an early version of this work.

References

[AAA+25] Marius A. Aardal, Gora Adj, Diego F. Aranha, Andrea Basso, Isaac Andrés Canales Martínez, Jorge Chavez-Saab, Maria Corte-Real Santos, Pierrick Dartois, Luca De Feo, Max Duparc, Jonathan Komada Eriksen, Tako Boris Fouotsa, Décio Luiz Gazzoni Filho, Basil Hess, David Kohel, Antonin Leroux, Patrick Longa, Luciano Maino, Michael Meyer, Kohei Nakagawa, Hiroshi Onuki, Lorenz Panny, Sikhar Patranabis, Christophe Petit, Giacomo Pope, Krijn Reijnders, Damien Robert, Francisco Rodríguez-Henríquez, Sina Schaeffler, and Benjamin Wesolowski. *SQIsign. Tech. rep., National Institute of Standards and Technology*, 2025. Available at https://csrc.nist.gov/Projects/pqc-dig-sig/round-2-additional-signatures.

[ACC+19] Gora Adj, Daniel Cervantes-Vázquez, Jesús-Javier Chi-Domínguez, Alfred Menezes, and Francisco Rodríguez-Henríquez. On the cost of computing isogenies between supersingular elliptic curves. In Carlos Cid and Michael J.: Jacobson, Jr., editors, *SAC 2018*, volume 11349 of *LNCS*, pages 322–343. Springer, Cham, August 2019. doi:10.1007/978-3-030-10970-7_15.

[BDD+24] Andrea Basso, Pierrick Dartois, Luca De Feo, Antonin Leroux, Luciano Maino, Giacomo Pope, Damien Robert, and Benjamin Wesolowski. SQIsign2D-west - the fast, the small, and the safer. In Chung and Sasaki [CS24], pages 339–370. doi:10.1007/978-981-96-0891-1_11.

[BDLS20] Daniel J. Bernstein, Luca De Feo, Antonin Leroux, and Benjamin Smith. Faster computation of isogenies of large prime degree. *ANTS XIV–Proceedings of the Fourteenth Algorithmic Number Theory Symposium*, 4(1):39–55, 2020. doi:10.2140/obs.2020.4.39.

[CH17] Craig Costello and Hüseyin Hisil. A simple and compact algorithm for SIDH with arbitrary degree isogenies. In Tsuyoshi Takagi and Thomas Peyrin, editors, *ASIACRYPT 2017, Part II*, volume 10625 of *LNCS*, pages 303–329. Springer, Cham, December 2017. doi:10.1007/978-3-319-70697-9_11.

[CLG09] Denis Xavier Charles, Kristin E. Lauter, and Eyal Z. Goren. Cryptographic hash functions from expander graphs. *Journal of Cryptology*, 22(1):93–113, January 2009. doi:10.1007/s00145-007-9002-x.

[Com26] Companion Repository for this Manuscript, 2026. The code is vailable at https://github.com/Crypto-TII/pqc-supersearcher.

[Cos20] Craig Costello. B-SIDH: Supersingular isogeny Diffie-Hellman using twisted torsion. In Shiho Moriai and Huaxiong Wang, editors, *ASIACRYPT 2020, Part II*, volume 12492 of *LNCS*, pages 440–463. Springer, Cham, December 2020. doi:10.1007/978-3-030-64834-3_15.

[CS24] Kai-Min Chung and Yu Sasaki, editors. *ASIACRYPT 2024, Part III*, volume 15486 of *LNCS*. Springer, Singapore, December 2024.

[CSCD+23] Jorge Chavez-Saab, Maria Corte-Real Santos, Luca De Feo, Jonathan Komada Eriksen, Basil Hess, David Kohel, Antonin Leroux, Patrick Longa, Michael Meyer, Lorenz Panny, Sikhar Patranabis, Christophe Petit, Francisco Rodríguez-Henríquez, Sina Schaeffler, and Benjamin Wesolowski. *SQIsign. Tech. rep., National Institute of Standards and Technology*, 2023. Available at https://csrc.nist.gov/Projects/pqc-dig-sig/round-1-additional-signatures.

[DF24] Max Duparc and Tako Boris Fouotsa. SQIPrime: A dimension 2 variant of SQISignHD with non-smooth challenge isogenies. In Chung and Sasaki [CS24], pages 396–429. doi:10.1007/978-981-96-0891-1_13.

[DG16] Christina Delfs and Steven D. Galbraith. Computing isogenies between supersingular elliptic curves over $\mathbb{F}_p$. *DCC*, 78(2):425–440, 2016. doi:10.1007/s10623-014-0010-1.

[DLRW24] Pierrick Dartois, Antonin Leroux, Damien Robert, and Benjamin Wesolowski. SQIsignHD: New dimensions in cryptography. In Joye and Leander [JL24], pages 3–32. doi:10.1007/978-3-031-58716-0_1.

[EHL+20] Kirsten Eisenträger, Sean Hallgren, Chris Leonardi, Travis Morrison, and Jennifer Park. Computing endomorphism rings of supersingular elliptic curves and connections to path-finding in isogeny graphs. *ANTS XIV–Proceedings of the Fourteenth Algorithmic Number Theory Symposium*, 4(1):215–232, 2020. doi:10.2140/obs.2020.4.215.

[JL24] Marc Joye and Gregor Leander, editors. *EUROCRYPT 2024, Part I*, volume 14651 of *LNCS*. Springer, Cham, May 2024.

[Koh96] David Kohel. *Endomorphism rings of elliptic curves over finite fields*. Phd thesis, University of California at Berkeley, 1996. Available at https://www.i2m.univ-amu.fr/perso/david.kohel/pub/thesis.pdf.

[Mon87] Peter L. Montgomery. Speeding the pollard and elliptic curve methods of factorization. *Mathematics of computation*, 48(177):243–264, 1987.

[Nat22] National Institute of Standards and Technology. *Call for Additional Digital Signature Schemes for the Post-Quantum Cryptography Standardization Process*, 2022. Available at https://csrc.nist.gov/csrc/media/Projects/pqc-dig-sig/documents/call-for-proposals-dig-sig-sept-2022.pdf.

[NOC+24] Kohei Nakagawa, Hiroshi Onuki, Wouter Castryck, Mingjie Chen, Riccardo Invernizzi, Gioella Lorenzon, and Frederik Vercauteren. SQIsign2D-east: A new signature scheme using 2-dimensional isogenies. In Chung and Sasaki [CS24], pages 272–303. doi:10.1007/978-981-96-0891-1_9.

[Ren18] Joost Renes. Computing isogenies between Montgomery curves using the action of (0, 0). In Tanja Lange and Rainer Steinwandt, editors, *Post-Quantum Cryptography - 9th International Conference, PQCrypto 2018*, pages 229–247. Springer, Cham, 2018. doi:10.1007/978-3-319-79063-3_11.

[SCS22] Maria Corte-Real Santos, Craig Costello, and Jia Shi. Accelerating the delfs-galbraith algorithm with fast subfield root detection. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part III*, volume 13509 of *LNCS*,

pages 285–314. Springer, Cham, August 2022. doi:10.1007/978-3-031-15982-4_10.

[SEMR24] Maria Corte-Real Santos, Jonathan Komada Eriksen, Michael Meyer, and Krijn Reijnders. AprèsSQI: Extra fast verification for SQIsign using extension-field signing. In Joye and Leander [JL24], pages 63–93. doi:10.1007/978-3-031-58716-0_3.

[The25] The Sage Developers. *SageMath, the Sage Mathematics Software System (Version 10.7)*, 2025. https://www.sagemath.org.

A Appendix

A.1 On the small tweak on $\mathsf{Searcher}_4$ and $\mathsf{SuperSearcher}_4$

Let us first focus on the algorithm $\mathsf{Searcher}_4$. Recall that the maximum number of subfield curves (that is, the number of visited nodes in the 4-isogeny tree) is approximately $\frac{4}{3} \cdot 4^h$ where $h = \log_4\left(\frac{\#\mathbb{S}_{p^2}}{\#\mathbb{S}_p}\right)$.

Let us follow the notation from Section 4, and let $\mathcal{E}_{\mathsf{b}}$ a node in the 4-isogeny tree rooted at (some given) supersingular curve $\mathcal{E}$. Observe that the optimized 4-isogeny formulas of [Ren18] exclude the explicit calculation of the intermediate 2-isogenies located between $\mathcal{E}_{\mathsf{b}}$ and $\mathcal{E}_{\mathsf{b}j}$ for $j \coloneqq 0, 1, 2, 3$. In particular, such intermediate 2-isogeny has as kernel generator the order-2 point $(0, 0)$ [Ren18].

There are two main remarks about such intermediate 2-isogenies:

1. The 4-isogenies $\phi_{\mathsf{b}0} \colon \mathcal{E}_{\mathsf{b}} \to \mathcal{E}_{\mathsf{b}0}$ and $\phi_{\mathsf{b}2} \colon \mathcal{E}_{\mathsf{b}} \to \mathcal{E}_{\mathsf{b}2}$ pass through the same intemerdiate 2-isogenous curve, and
2. The 4-isogenies $\phi_{\mathsf{b}1} \colon \mathcal{E}_{\mathsf{b}} \to \mathcal{E}_{\mathsf{b}1}$ and $\phi_{\mathsf{b}3} \colon \mathcal{E}_{\mathsf{b}} \to \mathcal{E}_{\mathsf{b}3}$ pass through the same intemerdiate 2-isogenous curve.

Then, the inspection of the intermediate curves through 2-isogenies with kernel $\langle (0, 0) \rangle$ includes

$$\sum_{k=0}^{h-1} 2 \cdot 4^k = \frac{2}{3} \cdot (4^h - 1) \approx \frac{2}{3} \cdot 4^h$$

extra subfield checks. Hence, we would inspect a total of $2 \cdot 4^h$ curves. Now, if $\mathcal{E}_{\mathsf{b}j} \colon y^2 = x^3 + Ax^2 + x$ for some $A = A'/C \in \mathbb{F}_{p^2}$ then the j-invariant of the intermediate 2-isogenous curves is

$$j = \frac{16 \cdot (A^2 + 12)^3}{(A^2 - 4)^2} = \frac{16 \cdot (A'^2 + 12C^2)^3}{C^2 \cdot (A'^2 - 4C)^2}.$$

Therefore, checking if $j \in \mathbb{F}_p$ requires $2\mathsf{M} + 4\mathsf{S} + 2\mathsf{mul} = 14\mathsf{mul}$ field operations when A' and C' are given. Hence, the inspection of the intermediate curves through 2-isogenies with kernel $\langle (0, 0) \rangle$ includes

$$\sum_{k=0}^{h-1} 2 \cdot 14 \cdot 4^k = \frac{28}{3} \cdot (4^h - 1) \approx \frac{28}{3} \cdot 4^h < 10 \cdot 4^h.$$

extra field operations over $\mathbb{F}_p$. Consequently, the total running time of $\mathsf{Searcher}_4$ is now bounded by $92 \cdot \left(\frac{\#\mathbb{S}_{p^2}}{\#\mathbb{S}_p}\right)$ field operations, while its average running time reduces to $23 \cdot \left(\frac{\#\mathbb{S}_{p^2}}{\#\mathbb{S}_p}\right)$ field operations.

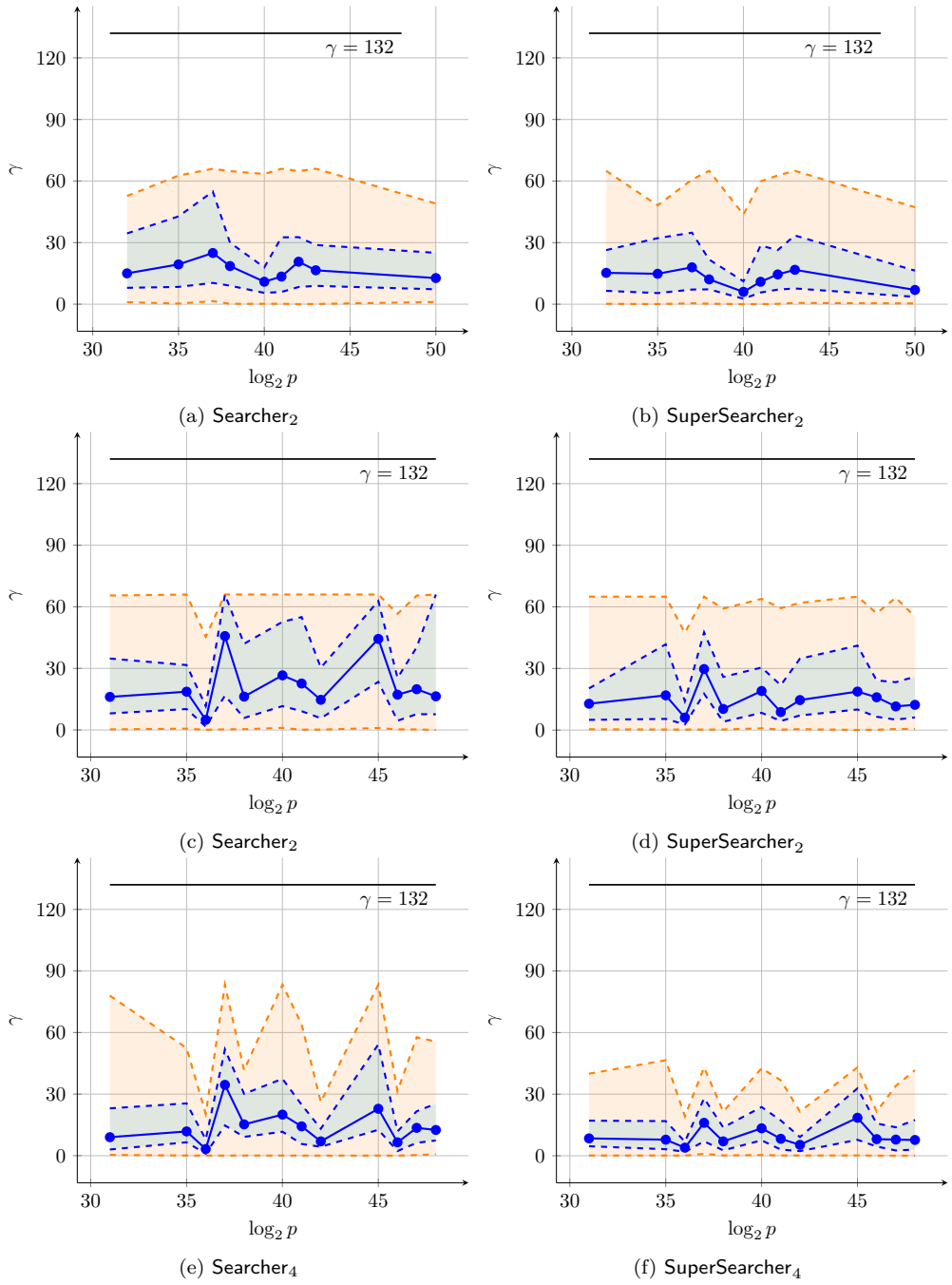
Let us now discuss the impact on the algorithm $\mathsf{SuperSearcher}_4$. In this scenario, it is just a matter of replacing 82 by 92 in the analysis of the running time of $\mathsf{SuperSearcher}_4$ (see Section 4.3). In particular, the running time of $\mathsf{SuperSearcher}_4$ changes to be bounded by $62.5 \cdot \left(\frac{\#\mathbb{S}_{p^2}}{\#\mathbb{S}_p}\right)$, while its average running time becomes $28.85 \cdot \left(\frac{\#\mathbb{S}_{p^2}}{\#\mathbb{S}_p}\right)$.

Hence, adding the 2-isogeny inspection only helps to slightly improve the average running time of $\mathsf{Searcher}_4$. In particular, it does not help to improve the algorithm $\mathsf{SuperSearcher}_4$.

A.2 Plots concerning the experiments

We graphically illustrate the comparison between our proposed algorithms in Figures 6a to 6f and 7a to 7d. The horizontal line in **black** ink determines the constant from Equation (1).

The dashed lines in **orange** ink describe the minimum and maximum values obtained from 25 random experiments, while the **blue** dashed lines refer to the first and third quartiles. The line in **blue** ink between the quartiles concerns the Median measured values.

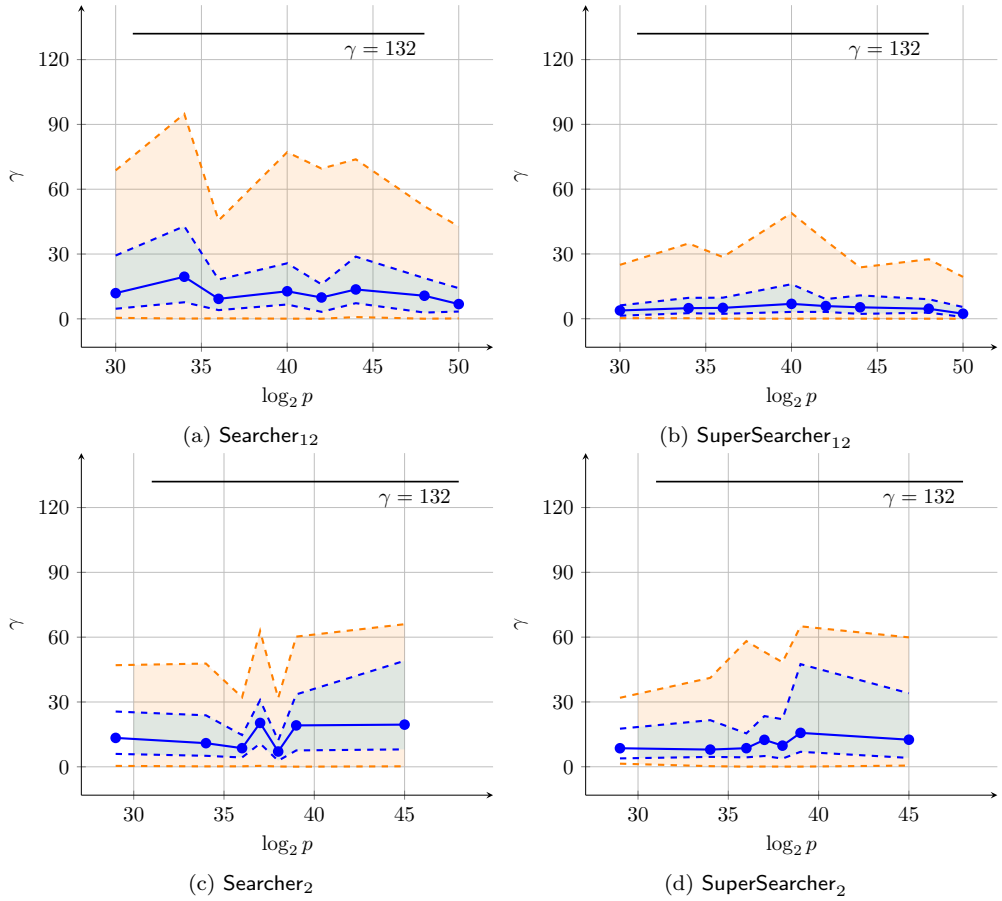


(a) Searcher$_2$ (b) SuperSearcher$_2$

(c) Searcher$_2$ (d) SuperSearcher$_2$

(e) Searcher$_4$ (f) SuperSearcher$_4$

Figure 6: Experiments in Figures 6a and 6b correspond with the prime numbers listed in Table 5. Experiments in Figures 6c and 6d correspond with the prime numbers listed in Table 6. Experiments in Figures 6e and 6f correspond with the prime numbers listed in Table 7.



(a) Searcher$_{12}$ (b) SuperSearcher$_{12}$

(c) Searcher$_2$ (d) SuperSearcher$_2$

Figure 7: Experiments in Figures 7a and 7b correspond with the prime numbers listed in Table 8. Experiments in Figures 7c and 7d correspond with the prime numbers listed in Table 9.

A.3 Parallel search optimizations.

Observe that the presented subfield curve searching algorithms are highly parallelizable when having $M = 2^m$ processors: one can precompute all the nodes at depth m, and each processor takes one of those nodes as the root and inspects a subtree of height $(d - m)$.

In what follows, we describe another interesting parallel search. Let us assume $M \in \mathbb{N}$, take an integer ℓ of $m = \lfloor \log_2 M \rceil$ bits, and proceed as below.

- Find a basis $\{R, S\}$ for $\mathcal{E}[\ell]$.
- For the kth processor:
  - Calculate the ℓ-isogeny $\psi_k \colon \mathcal{E} \to \mathcal{E}_k$ with kernel $\langle R + [k]S \rangle$.
  - Run the subfield curve searching algorithm with inputs $\mathcal{E}_k$, $\psi_k(P)$, $\psi_k(Q)$, and $h - m$.

The main limitation of the above parallel search is that we require ℓ to factorize as the product of "small" primes $\ell_1, \ldots, \ell_r$. But, for efficiency, most of the isogeny-based cryptographic constructions assume $(p+1)(p-1) = p^2 - 1$ is as smooth as possible (which favors this kind of parallel search). Therefore, one can take ℓ that divides $(p^2 - 1)$ and

satisfies $\ell \approx M$. [13] Notice that the cost of computing ψ_k is negligible compared with the running time of our subfield curve searching algorithms when $m \ll \frac{1}{2}\log_2 p$, and the number of processors is not limited to a power of two.

Certainly, the calendar time when applying the above parallel search techniques becomes $\mathcal{O}(p^{1/2}/M)$ but the total running time is still $\mathcal{O}(p^{1/2})$. [14]

A.4 Subfield Curve Searching Over More Generic Fields

At this point, we have provided subfield curve search algorithms with time complexity of

$$\mathcal{O}\left(\frac{\#\mathbb{S}_{p^2}}{\#\mathbb{S}_p}\right) = \mathcal{O}(p^{1/2})\ \mathsf{mul} \tag{6}$$

operations, for primes of the form $p = d^a \cdot f - 1$ for small $d \in \mathbb{N}$ and some $a, f \in \mathbb{N}$. Certainly, the condition in the form of p limits the applicability of the aforementioned algorithms $\mathsf{Searcher}_d$ and $\mathsf{SuperSearcher}_d$. However, we next discuss how to extend our current results to more generic primes.

Applicability to primes of the form $p = 2^a \cdot f + 1$. We emphasize that these algorithms straightforwardly apply to primes of the form $p = 2^a \cdot f + 1$. The reasoning for this claim relies on two facts: The torsion subgroup $\mathcal{E}[2^a]$ has two generators, P and Q, with x-coordinates in the field $\mathbb{F}_{p^2}$ and y-coordinates in $\mathbb{F}_{p^4}$ [Cos20], and the optimized Algorithms from Section 4 perform operations on the Kummer line (i.e., we only operate with the x-coordinates). The main difference when employing primes of the form $p = 2^a \cdot f + 1$ is that we work on a field extension $\mathbb{F}_{p^2} = \mathbb{F}_p[x]/(x^2 + \mu)$ with a small $\mu \in \mathbb{F}_p$ different from 1. That difference does not affect the asymptotic complexity of the algorithms (e.g., at the level of operations in $\mathbb{F}_{p^2}$, we still perform the same number of multiplications and squares).

Extending to other cryptographic interest primes. Notice that algorithm $\mathsf{Searcher}_d$ can be extended to the setting $p = 4 \cdot d_1 \cdot d_2 \cdots d_n - 1$ for some small odd primes d_i's. The main difference for that scenario relies on the isogeny step, which requires performing d_i-isogenies instead of d-isogenies. Each d_i-isogeny has a small degree $d_i \ll p^{1/2}$ and a computational cost of $\tilde{\mathcal{O}}(\sqrt{d_i})$ mul operations [BDLS20]. A similar situation occurs for a prime p such that $(p-1)$ and $(p+1)$ are B-smooth for some $B \in \mathbb{N}$ as in [Cos20, CSCD+23].

In both cases, the torsion subgroup $\mathcal{E}[p-1]$ has two generators, P and Q, with x-coordinates in the field $\mathbb{F}_{p^2}$ and y-coordinates in $\mathbb{F}_{p^4}$ [Cos20]. Hence, we can provide an algorithm that relies only on the x-coordinates and still allows a time complexity of $\mathcal{O}(p^{1/2})$ field multiplications in $\mathbb{F}_{p^2}$. The core idea relies on considering m-isogenies with $m \mid (p^2-1)$ such that $m = d_1 \cdot d_2 \cdots d_h \approx p^{1/2}$, $d_1 \leq d_2 \leq \cdots \leq d_h$ and the list $M = \{d_1, d_2, \ldots, d_h\}$ includes the h smallest prime factors of (p^2-1). Based on that, our proposed algorithm proceeds by computing d_i-isogenies going from d_h to d_1. Calculating the isogenies in that ordering ensures that the largest d-isogenies are performed the least number of times, while the smallest isogenies are computed as much as possible (e.g. an exponential number of times); thus, the running time is expected to be as in Equation (6) since the smaller isogenies have a running time close to $\mathcal{O}(1)$. We label $\mathsf{Searcher}_M$ to distinguish it from $\mathsf{Searcher}_d$, and summarize the main difference on this setting below.

[13] For the case of $\mathsf{SuperSearcher}_2$ and $\mathsf{RadicalSearcher}_2$, one should assume ℓ is not divisible by some integer in the input list L.

[14] By calendar time, we mean the elapsed time taken for a computation. The total time is the sum of the time expended by all M processors.

Setup when employing d_{n-1}-isogenies. Following the notation from Fig. 3, the input must satisfy $\mathcal{E}[m] = \langle P, Q\rangle$. In addition, the integer n determines the length of the sequence $\mathsf{b} \in \{0, \ldots, d_n - 1\} \times \{0, \ldots, d_{n-1} - 1\} \times \cdots \times \{0, \ldots, d_1 - 1\}$, and we denote $\mathbf{z}_\mathsf{b} = \{\mathcal{E}, \mathsf{b}, \mathrm{x}(P_\mathsf{b}), \mathrm{x}(Q_\mathsf{b}), \mathrm{x}(R_\mathsf{b})\}$ where $R_\mathsf{b} = P_\mathsf{b} - Q_\mathsf{b}$.

Calculating Codomain d_{n-1}-isogenies when $n < (h-1)$. We need to calculate d_{n-1} different d_{n-1}-isogenies:

$$\phi_{\mathsf{b}_j} : \mathcal{E}_\mathsf{b} \to \mathcal{E}_{\mathsf{b}_j} \text{ with } \ker \phi_{\mathsf{b}_j} = \langle [\prod_{i=1}^{n-1} d_i](P_\mathsf{b} + [j]Q_\mathsf{b})\rangle \text{ for each } j \coloneqq 0, \ldots, d_{n-1} - 1,$$

Calculating Codomain d_1-isogenies when $n = (h-1)$. We need to calculate d_1 different d_1-isogenies:

$$\phi_{\mathsf{b}_j} : \mathcal{E}_\mathsf{b} \to \mathcal{E}_{\mathsf{b}_j} \text{ with } \ker \phi_{\mathsf{b}_j} = \langle P_\mathsf{b} + [j]Q_\mathsf{b}\rangle \text{ for each } j \coloneqq 0, \ldots, d_1 - 1.$$

Pushing Points Through d_{n-1}-isogenies. We need to push a total of $3d_{n-1}$ x-only points through the isogenies:

$$\begin{aligned} &\mathrm{x}(P_{\mathsf{b}_j}) = \mathrm{x}(\phi_{\mathsf{b}_j}(P_\mathsf{b} + [j]Q_\mathsf{b})), && \mathrm{x}(Q_{\mathsf{b}_j}) = \mathrm{x}(\phi_{\mathsf{b}_j}([d_{n-1}]Q_\mathsf{b})), \text{ and} \\ &\mathrm{x}(R_{\mathsf{b}_j}) = \mathrm{x}(\phi_{\mathsf{b}_j}(P_\mathsf{b} - [d_{n-1} - j]Q_\mathsf{b})) \end{aligned}$$

for each $j \coloneqq 0, \ldots, d_{n-1} - 1$.

Remark 4. Since the subfield curve detection algorithm in Section 4.3 exploits inspection through small d-isogenies, it suggests that a variant of $\mathsf{SuperSearcher}_M$ must require to inspect for values of $\ell \notin M$. In particular, for primes of the form $p = 4 \cdot d_1 \cdot d_2 \cdots d_n - 1$, we expect that $\mathsf{SuperSearcher}_M$ coincides with $\mathsf{Searcher}_M$ (i.e., $L = \emptyset$). A similar situation should occur for primes p such that $(p-1)$ and $(p+1)$ are B-smooth.