Improved Subfield Curve Search For Specific Field Characteristics

Jesús-Javier Chi-Domínguez

Technology Innovation Institute, Abu Dhabi, UAE
jesus.dominguez@tii.ae

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.
The Delfs-Galbraith algorithm is the most efficient procedure 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 base field), which determines the time complexity.
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 focuses on primes of that particular form, and it presents two new algorithms for finding subfield curves with a time complexity of $\mathcal{O}(p^{1/2})$ operations and a memory complexity polynomial in $\log_2 p$. Such algorithms exploit the existence of large torsion-2^a points and extend the subfield root detection algorithm of Santos, Costello, and Shi (Crypto 2022) to our case study. In addition, it is worth highlighting that these algorithms easily extend to primes of the form $p = 2^a \cdot f + 1$ and $p = \ell^a \cdot f - 1$ with ℓ being a small integer.
This study also examines the usage of radical 3-isogenies with the proposed extended subfield root detection algorithm. In this context, the results indicate that the radical 3-isogeny approach is competitive compared with the state-of-the-art algorithms.

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 [13], 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.

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

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.

In addition, 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 isogeny calculations performance, and it does not affect security, recent works push for primes of the form $p = 2^a \cdot f - 1$. For example,

- [7] requires primes $p = (4 \cdot 3)^a \cdot f - 1$ with $f \in \mathbb{N}$ and $a \approx \frac{1}{2} \log_{12} p$.
- [12] 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 $b \approx \frac{1}{2} \log_3 p$.
- [24] works over primes of the form $p = 2^a \cdot f - 1$ with $f \in \mathbb{N}$ and $a \approx \frac{1}{2} \log_2 p$.
- [2,20,14,19] asks for $p = 2^a \cdot f - 1$ with small $f \in \mathbb{N}$ and $a \approx \log_2 p$.

In addition, all the above prime numbers determine an interesting structure in the torsion-$(p+1)$ subgroup of the curves. 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 torsion-2^a 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 DFS 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.

Secondly, we extend the subfield root detection algorithm from [23] 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 [23] 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 $\log_2 p$. Additionally, we provide a tighter complexity analysis of the algorithm in [23], which includes the hidden logarithm factor in the complexity analysis from [23]. Furthermore, we present another algorithm, namely $\mathsf{RadicalSearcher}_3$, that combines our proposed subfield root detection with radical 3-isogenies for finding subfield curves over more general primes. 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 [26] available at [21].

Algorithm	**Running Time**	
	Average	**Maximum**
SuperSolver	$c_1 \cdot \log_2 p \cdot 2^h$	$c_2 \cdot \log_2 p \cdot 2^h$
Searcher	$33 \cdot 2^h$	$66 \cdot 2^h$
$\mathsf{Searcher}_4$	$33 \cdot 2^h$	$88 \cdot 2^h$
$\mathsf{Searcher}_{12}$	$58.21 \cdot 2^h$	$127 \cdot 2^h$
SuperSearcher	$20 \cdot 2^h$	$65 \cdot 2^h$
$\mathsf{SuperSearcher}_4$	$17.45 \cdot 2^h$	$32 \cdot 2^h$
$\mathsf{SuperSearcher}_{12}$	$27.68 \cdot 2^h$	$226.49 \cdot 2^h$
$\mathsf{RadicalSearcher}_3$	$c_3 \cdot \log_2 p \cdot 2^h$	$c_4 \cdot \log_2 p \cdot 2^h$

Table 1. The constants $0.2 \lesssim c_1 \lesssim 1.64$ and $0.07 \lesssim c_2 \lesssim 0.51$ are as detailed in Section 3, while constants $0.14 \lesssim c_3 \lesssim 1.44$ and $0.08 \lesssim c_4 \lesssim 0.62$ are as determined in Section 5. The row concerning SuperSearcher corresponds with our analysis on the algorithm SuperSolver from [23]. The exponent value of h is $\frac{\#\mathbb{S}_{p^2}}{\#\mathbb{S}_p}$.

We emphasize that our results should be taken as cautionary advice for proposing 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 $\lambda \in \{128, 192, 256\}$. In addition, our work easily extends to primes of the form $p = 2^a \cdot f + 1$ and $p = \ell^a \cdot f - 1$.

Organization. Section 2 provides the preliminaries. Section 3 details and refines an analysis of the running time of SuperSolver. Section 4 describes the two algorithms, $\mathsf{Searcher}_d$ and $\mathsf{SuperSearcher}_d$ with special focus on $d = 2, 4, 12$; in particular, we present the second algorithm in Section 4.2. We combine our results from Section 4.2 with the usage of radical isogenies in Section 5: we show that using radical 3-isogenies becomes competitive against SuperSolver. We present all our experiments in Section 6 and discuss concluding remarks of our work in Section 7.

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]

Let $p = 2^a f - 1$ be a prime integer 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 $j(\mathcal{E}) = j(\mathcal{F})$. In addition, we use the term "subfield curve" to mean $j(\mathcal{E})$ 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. $\mathcal{F}$). All the isogenies in this work are assumed to have cyclic kernels. We say that ϕ is an ℓ-isogeny if $\ker\phi$ 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. There are exactly $(\ell+1)$ ℓ-isogenies (up to isomorphism) with domain $\mathcal{F}$, one of them corresponding with $\hat{\phi}\colon \mathcal{F} \to \mathcal{E}$.

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 [16], and it is an expander graph [6]. 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 $\#\mathbb{S}_p = \mathcal{O}(p^{1/2})$ of them belong to $\mathbb{F}_p$ [13].

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 $4\mathsf{M} + 2\mathsf{S} + 4\mathsf{A}$

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

operations, respectively [17]. In addition, computing $\mathrm{x}([3]P)$ requires $7\mathsf{M}+5\mathsf{S}+10\mathsf{A}$.

Similarly, ℓ-isogenies on $\mathcal{E}$ nicely restrict to $\mathcal{E}/\langle\ominus\rangle$ [11,22,3]; 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 $\ell=2,3,4$.

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

Table 2. The 2-isogeny and 4-isogeny formulas from [22] assume isogeny kernels different from $(0,0)$. The 3-isogeny cost comes from [11].

3 On the complexity of Delfs-Galbraith Subfield Searching Algorithm

Santos, Costello, and Shi provide an efficient procedure for determining whether an ℓ-isogeny connects a given elliptic curve to a subfield curve [23]. 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 $\mathsf{NeighbourInFp}(\ell, j)$, which determines whether the polynomial $\Phi_\ell(x,j)$ has roots in the field $\mathbb{F}_p$; we give a high-level description of $\mathsf{NeighbourInFp}$ in Figure 1. The main ingredients of $\mathsf{NeighbourInFp}$ include i) a free-inverse 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 $\mathsf{EvaluateModularPolynomial}(\ell, j)$ is bounded by $9d_\ell(d_\ell-1)\mathsf{mul}$ operations, while $\mathsf{InverseFreeGCD}$ by

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

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

3.1 Overall description of SuperSolver algorithm

As the main application of $\mathsf{NeighbourInFp}$, Santos, Costello, and Shi propose using it in the Delfs-Galbraith Subfield Searching Algorithm. Following [23] notation, below we give a high-level description of such an optimized subfield-searching algorithm, $\mathsf{SuperSolver}$.

[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.

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

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

Fig. 1. Description of the subfield root detection algorithm from [23]. The input 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 [23]: it evaluates $\Phi_\ell(x, y)$ at $y = j$ by calculating $j^2, \ldots, y^{d_\ell}$ and multiplying them by the corresponding coefficients of $\Phi_\ell(x, y)$.

Criteria for successfully reaching subfield curves. Given that the supersingular isogeny graph $\mathcal{X}(\overline{\mathbb{F}}_p, 2)$ is an expander graph (in particular, there is a short isogeny path between any two supersingular curves), and that the number of supersingular elliptic curves defined over $\mathbb{F}_{p^2}$ is about $\frac{1}{12}p$, with only $\tilde{\mathcal{O}}(p^{1/2})$ subfield curves. Then, a short 2-isogeny path of length

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

from $\mathcal{E}$ to a subfield curve $\mathcal{F}$ should exist [13]. Consequently, the time complexity of finding a subfield curve through algorithm SuperSolver is $\tilde{\mathcal{O}}(p^{1/2})$ field operations with a memory complexity polynomial in $\log_2 p$.

3.2 On the upper bound and expected average cost of SuperSolver

To our knowledge, [23] only provides an experimental average cost for SuperSolver of

$$58 \cdot \left(\frac{\#\mathbb{S}_{p^2}}{\#\mathbb{S}_p}\right)$$

field operations. However, as already highlighted, there is a hidden logarithm factor in the complexity of the Delfs-Galbraith Subfield Searching Algorithm determined by Line 13 in SuperSolver, which requires a square root calculation. Given that, we next describe an explicit upper bound on the cost of SuperSolver, including the logarithmic factor (this explicit runtime will be the baseline for comparing our proposed algorithms in Section 4).

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

Fig. 2. Description of the optimized subfield curve search algorithm from [23]. The input must satisfy $\mathcal{E}$ and $\mathcal{F}$ are connected by a 2-isogeny. 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 $n = 0$. 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.

Cost metric for NeighbourInFp*:* We use the subfield cost for Line 8 in the SuperSolver algorithm as in [23]; that is, we take that cost (per ℓ) as

$$\mathsf{cost}_\ell = \frac{1}{d_\ell}(\text{cost of } \mathsf{NeighbourInFp}) \leq \frac{1}{d_\ell}(11d_\ell^2 - 6d_\ell - 5)\mathsf{mul}.$$

Cost metric for solving $\frac{\Phi_2(j(\mathcal{E}_\mathsf{b}),x)}{x-j(\mathcal{F}_\mathsf{b})}$*:* Line 13 in the SuperSolver algorithm requires solving a quadratic equation, which summarizes as calculating

$$j(\mathcal{E}_{\mathsf{b}i}) = \frac{1}{2}\left(j(\mathcal{E}_\mathsf{b})^2 - 1488j(\mathcal{E}_\mathsf{b}) - j(\mathcal{F}_\mathsf{b}) + 162000 \pm \sqrt{\rho_i}\right) \qquad (2)$$

where

$$\rho_i = j(\mathcal{E}_{\mathsf{b}})^4 - 2976j(\mathcal{E}_{\mathsf{b}})^3 + 2j(\mathcal{E}_{\mathsf{b}})^2 j(\mathcal{F}_{\mathsf{b}}) + 2532192j(\mathcal{E}_{\mathsf{b}})^2 - 2976j(\mathcal{E}_{\mathsf{b}})j(\mathcal{F}_{\mathsf{b}}) - 645205500j(\mathcal{E}_{\mathsf{b}}) - 3j(\mathcal{F}_{\mathsf{b}})^2 + 324000j(\mathcal{F}_{\mathsf{b}}) - 8748000000.$$

Using Scott's fast square root method [25], we observe that calculating $\sqrt{\rho_i}$ takes two multiplications and two exponentiations over $\mathbb{F}_p$; in particular, it requires raising to the powers of

$$N = \frac{1}{4}(p+1) \quad \text{and} \quad M = \frac{1}{4}(p-3).$$

In practice, for concrete cryptographic parameters, both N and M have approximately $\frac{1}{2}(r-2)$ as Hamming weight, respectively. For simplicity, we conservatively opt to use the Left-to-right binary algorithm for the field exponentiation. We highlight that we could further optimize such exponentiation by employing the wNAF method or a shorter addition chain. However, this improvement would have only a tiny impact on our purposes and comparisons. Hence,

– Calculating $z^{(p+1)/4}$ takes $\frac{1}{2}(r-2)\mathsf{mul} + (r-2)\mathsf{sqr}$ operations, and
– Computing $z^{(p-1)/3}$ requires $\frac{1}{2}(r-2)\mathsf{mul} + (r-2)\mathsf{sqr}$ operations.

Therefore, the cost of calculating Equation (2) becomes

$$\begin{aligned}\mathsf{cost}_{\mathsf{eqn}} &= (2\mathsf{mul} + 3\mathsf{sqr}) + (r-2)\mathsf{mul} + 2(r-2)\mathsf{sqr} + (3\mathsf{M} + 3\mathsf{S}) \\ &= (r+15)\mathsf{mul} + (2r-1)\mathsf{sqr} \approx (3r+14)\mathsf{mul}.\end{aligned} \tag{3}$$

On the explicit running time of SuperSolver: We describe below a different analysis approach from [23], enabling us to isolate the constant factor in the complexity of the algorithm SuperSolver, thus allowing us to describe the running time of SuperSolver explicitly.

Recall, NeighbourInFp allows us to detect if there is one ℓ-isogenous subfield curve among the $(\ell+1)$ possibilities. That optimization implies an increased number of inspected curves by a factor of $(1 + \sum_{\ell \in L} d_\ell)$ times bigger. Therefore, we can decrease the value of h to (at most)

$$h = \log_2\left(\frac{\#\mathbb{S}_{p^2}}{\#\mathbb{S}_p}\right) - \left\lfloor \log_2\left(1 + \sum_{\ell \in L} d_\ell\right) \right\rceil$$

and still inspect around $\frac{\#\mathbb{S}_{p^2}}{\#\mathbb{S}_p}$ curves. We thus consider the running time of algorithm SuperSolver to be bounded by

$$\mathsf{runtime}_1 = \underbrace{\frac{1}{2^{\left\lfloor \log_2\left(1+\sum_{\ell\in L} d_\ell\right)\right\rceil}}\left(\mathsf{cost}_{\mathsf{eqn}} + 2 \cdot \sum_{\ell \in L} \overbrace{d_\ell \cdot \mathsf{cost}_\ell}^{\mathsf{NeighbourInFp}}\right)}_{\gamma_p(L)} \cdot 2^h \;\mathsf{mul}. \tag{4}$$

We take advantage of the following three main observations related to function $\gamma_p(L)$:

1. $\sum_{\ell\in L} d_\ell$ is linear in the variables $\ell_1,\ldots,\ell_t$.
2. $\sum_{\ell\in L} d_\ell \cdot \mathsf{cost}_\ell$ is quadratic in the variables $\ell_1,\ldots,\ell_t$.
3. Function $\gamma_p(L)$ is polynomial in $\ell_1,\ldots,\ell_t$, and $\log_2 p$.

Given that the exponential factor

$$2^h \approx \frac{\#\mathbb{S}_{p^2}}{\#\mathbb{S}_p}$$

in the running time does not change from the choice of L, we focus on minimizing the value of $\frac{1}{\log_2 p}\cdot\gamma_p(L) \leq \frac{\mathsf{cost}_{\mathsf{eqn}}}{\log_2 p} \in O(1)$. In particular, we use the concrete costs of cost_ℓ from [23] 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

$$\underset{\mathbf{b}}{\operatorname{argmin}}\ \frac{1}{\log_2 p}\cdot\gamma_p(L_{\mathbf{b}}),$$

where $L_{\mathbf{b}}$ denotes the list having the i^{th} odd prime ℓ_i if the i^{th} bit of $\mathbf{b}$ is equal to one. Our experiments illustrate that the running time of $\mathsf{SuperSolver}$ is bounded by

$$\mathsf{runtime}_1 = c\cdot\log_2 p\cdot 2^h\ \mathsf{mul} = c\cdot\log_2 p\cdot\left(\frac{\#\mathbb{S}_{p^2}}{\#\mathbb{S}_p}\right)\mathsf{mul} \tag{5}$$

where $0 < c \lesssim 1.64$.

On the expected average running time of $\mathsf{SuperSolver}$: Observe that the algorithm $\mathsf{SuperSolver}$ inspects

$$2^{h+1}\cdot(1+\sum_{\ell\in L} d_\ell)$$

curves, where $h = \log_2\left(\frac{\#\mathbb{S}_{p^2}}{\#\mathbb{S}_p}\right) - \lfloor\log_2(1+\sum_{\ell\in L} d_\ell)\rceil$.

Remark 1. Notice that, in the best-case scenario, the input curve is a subfield curve, while in the worst-case scenario, we inspect all the curves. Therefore, on average, one would expect to inspect half of the curves to reach a subfield curve.

Hence, given that we only need to inspect at most $\frac{\#\mathbb{S}_{p^2}}{\#\mathbb{S}_p}$ curves to find a subfield curve, the expected running time of the algorithm $\mathsf{SuperSolver}$ turns out to be

$$\frac{1}{2}\cdot\left(\frac{2^{\lfloor\log_2(1+\sum_{\ell\in L} d_\ell)\rceil}}{2\cdot(1+\sum_{\ell\in L} d_\ell)}\cdot\mathsf{runtime}_1\right) = \left(\frac{2^{\lfloor\log_2(1+\sum_{\ell\in L} d_\ell)\rceil}}{4\cdot(1+\sum_{\ell\in L} d_\ell)}\cdot\mathsf{runtime}_1\right).$$

For example, for L_5 we have $0.31\cdot\mathsf{runtime}_1$, while for L_{11} we obtain $0.35\cdot\mathsf{runtime}_1$. In particular, when p increases, the constant c decreases slightly (see Table 3).

$\log_p$	32	64	96	128	192	256	384	512
c (max.)	1.64	0.91	0.67	0.53	0.38	0.31	0.24	0.20
c (avg.)	0.51	0.28	0.21	0.16	0.13	0.11	0.08	0.07
L	L_5	L_5	L_5	L_5	L_{11}	L_{11}	L_{11}	L_{11}

Table 3. The row corresponding to L presents the best choice of L. In particular, we have $L_5 = [3, 7]$ and $L_{11} = [3, 5, 11]$. The row corresponding with c (max.) determines the constant factor in the running time of SuperSolver (i.e., Equation (5)), while c (avg.) with the average running time of SuperSolver.

4 Searcher and SuperSearcher: Efficient Subfield Curve Searching Over Specific Field Characteristics

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 [1].

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.

Time complexity: 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 Line 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, the cost of inspecting the 2-isogeny subtree is approximately $2 \cdot 2^h$ subfield checks, $\frac{5}{2} \cdot 2^h$ point doublings, 2^h point additions, $2 \cdot 2^h$ isogeny evaluations, and $2 \cdot 2^h$ isogeny codomain calculations. In particular, we get an asymptotical running time of $\mathcal{O}(2^h)$ field operations since each point and isogeny operation takes $\mathcal{O}(1)$ field operations.

Memory complexity: 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 $(X\colon Z)$.
2. The node $\mathbf{z}_{\mathsf{b}}$ has an extra point, $\mathbf{z}_{\mathsf{b}} = \{\mathcal{E}, \mathsf{b}, \mathrm{x}(P_{\mathsf{b}}), \mathrm{x}(Q_{\mathsf{b}}), \mathrm{x}(R_{\mathsf{b}})\}$ where $R_{\mathsf{b}} = P_{\mathsf{b}} - Q_{\mathsf{b}}$.
3. The elliptic curve in $\mathbf{z}_{\mathsf{b}}$ is represented by its projective representation, $(A' + 2C\colon 4C)$ where $A = A'/C$.

```
Searcher($\mathcal{E}, P, Q, h$)
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 $\ker \phi_{\mathsf{b}0} = \langle [2^{h-n-1}](P_\mathsf{b} + Q_\mathsf{b}) \rangle$
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 return
23 : if S is empty
24 :    return $\perp$
25 : else
26 :    Go to Line 3
27 : endif
```

Fig. 3. Description of our subfield curve seacrh algorithm. The input must satisfy $\mathcal{E}[2^h] = \langle P, Q \rangle$. The integer n determines the length of the bitstring b. For the empty bitstring $\mathsf{b} = \varnothing$, we set $n = 0$. 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 $\ker \widehat{\phi}_{\mathsf{b}i} = \langle [2^{h-n-2}] Q_{\mathsf{b}i} \rangle$.

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 $\mathrm{x}(R_{\mathsf{b}1}) = \mathrm{x}(\phi_{\mathsf{b}1}(P_\mathsf{b} - Q_\mathsf{b}))$.

Hence, the cost of inspecting the 2-isogeny subtree slightly changes as approximately $2 \cdot 2^h$ subfield checks, $\frac{5}{2} \cdot 2^h$ point doublings, 2^h point additions, $3 \cdot 2^h$ isogeny evaluations, and $2 \cdot 2^h$ isogeny codomain calculations. Since one can perform a subfield check with $2\mathsf{M} + 4\mathsf{S} + 16\mathsf{A} + 2\mathsf{mul}$ operations, [4] we get a running time bounded by

$$\mathsf{runtime}_2 = (30\mathsf{M} + 19\mathsf{S} + 4\mathsf{mul})2^h = 132 \cdot 2^h \ \mathsf{mul}.$$

Now, observe that the algorithm $\mathsf{Searcher}$ inspects $2^{h+1} - 1$ curves, doubling the required curves. 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 $\mathsf{Searcher}$ is bounded by

$$\mathsf{runtime}_2 = 132 \cdot 2^h \ \mathsf{mul} = \frac{132}{2} \cdot \left(\frac{\#\mathbb{S}_{p^2}}{\#\mathbb{S}_p}\right) \mathsf{mul} = 66 \cdot \left(\frac{\#\mathbb{S}_{p^2}}{\#\mathbb{S}_p}\right) \mathsf{mul}. \quad (6)$$

In particular, following Remark 1,the expected *average* running time of $\mathsf{Searcher}$ becomes

$$\frac{1}{2} \cdot \mathsf{runtime}_2 = 33 \cdot \left(\frac{\#\mathbb{S}_{p^2}}{\#\mathbb{S}_p}\right) \mathsf{mul}.$$

The Impact of Using 4-isogenies in Searcher. Computing 4-isogenies is much cheaper than composing two consecutive 2-isogenies (this is already one optimization trick used in many isogeny cryptographic implementations). 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 $\mathsf{Searcher}$, we denote the 4-variant algorithm by $\mathsf{Searcher}_4$.

Setup when employing 4-isogenies. Following the notation from Figure 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.

[4] 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 $ad = bc$.

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.

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

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

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 $\mathrm{x}(R_{\mathsf{b}3})$.

On the running time of $\mathsf{Searcher}_4$*:* The cost of inspecting the 4-isogeny subtree slightly changes as approximately $\frac{4}{3} \cdot 4^h$ subfield checks, $\frac{2}{3} \cdot 4^h$ point doublings, $\frac{17}{12} \cdot 4^h$ point additions, 4^h isogeny evaluations, and $\frac{4}{3} \cdot 4^h$ isogeny codomain calculations. Consequently, we get a running time bounded by

$$\mathsf{runtime}_4 = \frac{1048}{12} \cdot 4^h \ \mathsf{mul} < 88 \cdot 2^d \ \mathsf{mul} = 88 \cdot \left(\frac{\#\mathbb{S}_{p^2}}{\#\mathbb{S}_p}\right) \mathsf{mul}. \tag{7}$$

Remark 2. Differently from the algorithm $\mathsf{Searcher}$, 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 lesser than $\frac{\#\mathbb{S}_{p^2}}{\#\mathbb{S}_p}$. Therefore, we set

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

for $\mathsf{Searcher}_4$, which implies inspecting $\frac{4}{3}\left(\frac{\#\mathbb{S}_{p^2}}{\#\mathbb{S}_p}\right)$ curves. Hence, from Remark 1, the expected *average* running time of $\mathsf{Searcher}_4$ turns out to be

$$\frac{1}{2} \cdot \left(\frac{3}{4} \cdot \mathsf{runtime}_4\right) = 33 \cdot \left(\frac{\#\mathbb{S}_{p^2}}{\#\mathbb{S}_p}\right) \mathsf{mul}.$$

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

It is worth noting that the algorithm $\mathsf{Searcher}$ 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 Figure 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 [7,12]). Hence, let us take $d = 12$. This time, the cost of inspecting the d-isogeny subtree slightly changes as approximately $\frac{12}{11} \cdot 12^h$ subfield checks, $\frac{3}{121} \cdot 12^h$ point multiplications by twelve, $\frac{51}{44} \cdot 12^h$ point additions, $\frac{3}{11} \cdot 12^h$ isogeny evaluations, and $\frac{12}{11} \cdot 12^h$ isogeny codomain calculations. Now, notice that

- A point multiplication by twelve requires calculating two-point doubling and one-point tripling.

– 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 trinpling, one 4-isogeny curve codomain calculation, one 4-isogeny evaluation, and one last 3-isogeny curve codomain calculation.
  - Pushing a point through a 12-isogeny implies pushing the point through a 4-isogeny and a 3-isogeny.

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

$$\mathsf{runtime}_{12} = \frac{61452}{484} \cdot 12^h \ \mathsf{mul} < 127 \cdot 12^h \ \mathsf{mul} = 127 \cdot \left(\frac{\#\mathbb{S}_{p^2}}{\#\mathbb{S}_p}\right) \ \mathsf{mul}. \tag{8}$$

Furthermore, the *expected* average running time of $\mathsf{Searcher}_{12}$ becomes

$$\frac{1}{2}\left(\frac{11}{12} \cdot \mathsf{runtime}_{12}\right) \approx 58.21 \cdot \left(\frac{\#\mathbb{S}_{p^2}}{\#\mathbb{S}_p}\right) \ \mathsf{mul}.$$

4.2 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 [23] without the calculation of v^{-1}. [5] 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

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

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

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

[5] Santos, Costello, and Shi highlight that having the j-invariants explicitly computed is critical in their subfield root detection algorithm [23, Remark 2].

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

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

Observe that $\gcd(g_1, g_2) = \gcd(w \cdot g_1, w \cdot g_2)$ holds, and 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 $\mathbb{F}_p$. [6] Thus, our modified subfield root detection algorithm ScaledNeighbourInFp requires at most $(11d_\ell^2 + 3d_\ell - 14)$mul operations. We give a high-level description of ScaledNeighbourInFp in Figure 4.

```
ScaledNeighbourInFp($\ell, u, v$)
1 :  $f^\star \leftarrow$ EvaluateScaledModularPolynomial($\ell, u, v$)
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($g_1^\star, g_2^\star$)
```

Fig. 4. Description of our modified subfield root detection algorithm. The input must satisfies that $j(\mathcal{E}) = \frac{u}{v}$ for some supersingular elliptic 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.

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

[6] 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.

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.

Optimizing the choice of each $\boldsymbol{\ell_k}$. The situation here differs from [23] 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 $\mathsf{ScaledNeighbourInFp}$, and try to 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 [23] to better understand the difference in cost of inspecting $\mathcal{X}(\overline{\mathbb{F}}_p, \ell)$ when employing $\mathsf{NeighbourInFp}$ and $\mathsf{ScaledNeighbourInFp}$; more precisely, we take that cost as

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

Table 4 presents and compares the concrete cost for inspecting $\mathcal{X}(\overline{\mathbb{F}}_p, \ell)$ when employing $\mathsf{NeighbourInFp}$ and $\mathsf{ScaledNeighbourInFp}$. Observe that the procedure $\mathsf{ScaledNeighbourInFp}$ is slightly costlier than $\mathsf{NeighbourInFp}$, but we highlight that the algorithm $\mathsf{Searcher}_d$ with $\mathsf{NeighbourInFp}$ adds logarithmic factors in the overall running time while with $\mathsf{ScaledNeighbourInFp}$ does not.

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
$\mathsf{cost}_\ell^\star$:	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

Table 4. The row corresponding to cost_ℓ presents the values from [23, Table 3].

We thus consider the running time of algorithm $\mathsf{SuperSearcher}_d$ to be bounded by

$$\mathsf{runtime}_\mathsf{d}^* = \underbrace{\frac{1}{d^{\left\lfloor \log_d \left(1 + \sum_{\ell \in L} d_\ell\right) \right\rceil}} \left(\mathsf{c} + \frac{d}{d-1} \cdot \sum_{\ell \in L} \overbrace{d_\ell \cdot \mathsf{cost}_\ell^*}^{\mathsf{ScaledNeighbourInFp}} \right)}_{\gamma(L)} \cdot d^h\ \mathsf{mul}, \qquad (9)$$

where

$$\mathsf{c} = \begin{cases} 132 & \text{if } d = 2, \\ 88 & \text{if } d = 4, \\ 127 & \text{if } d = 12. \end{cases}$$

Observe that $\gamma(L)$ does not depend on either $\log_2 p$ nor d; therefore, we can consider it a constant factor. With that in mind, we want to minimize the value of $\gamma(L) \leq 132$. We combine the concrete costs from Table 4 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

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

where $L_{\mathbf{b}}$ denotes the list having the i^{th} odd prime ℓ_i if the i^{th} bit of $\mathbf{b}$ is equal to one. 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

– $L_5 = [3, 7]$ with $\gamma(L_5) = 65$ when employing the algorithm $\mathsf{SuperSearcher}_2$,
– $L_3 = [3, 5]$ with $\gamma(L_3) = 32$ when performing the algorithm $\mathsf{SuperSearcher}_4$,
– $L_{31} = [3, 5, 7, 11, 13]$ with $\gamma(L_{31}) = 18.874368$ when using $\mathsf{SuperSearcher}_{12}$.

Consequently, we get that the algorithm $\mathsf{SuperSearcher}_d$ has a running time bounded by

$$\mathsf{runtime}^*_{\mathsf{d}} = \gamma \cdot 2^h\ \mathsf{mul} = \gamma \cdot \left(\frac{\#\mathbb{S}_{p^2}}{\#\mathbb{S}_p}\right) \mathsf{mul} \tag{10}$$

with

$$\gamma = \begin{cases} 65 & \text{if } d = 2, \\ 32 & \text{if } d = 4. \end{cases}$$

For the case $d = 12$, the number of inspected curves is approximately $0.45 \cdot 2^h$; thus, we need to increase the h to $\log_d\left(\frac{\#\mathbb{S}_{p^2}}{\#\mathbb{S}_p}\right) - \lfloor \log_d (1 + \sum_{\ell \in L} d_\ell) \rceil + 1$, which gives $\gamma = 18.874368 \cdot 12 = 226.492416$

On the average running time of $\mathsf{SuperSearcher}_d$*:* Observe that the algorithm $\mathsf{SuperSearcher}_d$ inspects

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

curves, where $h = \log_d\left(\frac{\#\mathbb{S}_{p^2}}{\#\mathbb{S}_p}\right) - \lfloor \log_d (1 + \sum_{\ell \in L} d_\ell) \rceil$. Hence, according to Remark 1, the expected *average* running time of the algorithm $\mathsf{SuperSearcher}_d$ turns out to be [7]

$$\frac{1}{2} \cdot \left(\frac{(d-1) \cdot d^{\lfloor \log_d (1 + \sum_{\ell \in L} d_\ell) \rceil}}{d \cdot (1 + \sum_{\ell \in L} d_\ell)} \cdot \mathsf{runtime}^*_{\mathsf{d}} \right) = \mathsf{constant} \cdot \left(\frac{\#\mathbb{S}_{p^2}}{\#\mathbb{S}_p}\right) \mathsf{mul},$$

[7] For the case $d = 12$, we need to divide by $d = 12$ since we have an increased value of $h \leftarrow h + 1$.

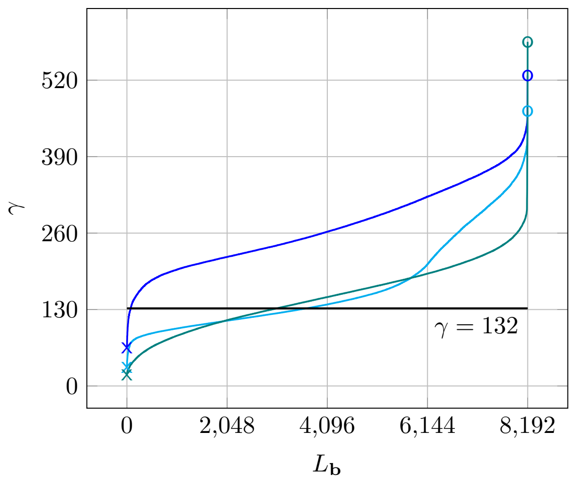


Fig. 5. Value of $\gamma(L_{\mathbf{b}})$ for each $\mathbf{b} \in \{0,1\}^{13}$. The x-axis represents the choice of the list $L_{\mathbf{b}}$ sorted from the one with minimal constant $\min \gamma(L_{\mathbf{b}})$ to the one with maximal constant $\max \gamma(L_{\mathbf{b}})$. The y-axis determines the constant $\gamma(L_{\mathbf{b}})$ as given in Equation (9). In addition, 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 determines the values for algorithm SuperSearcher$_4$ (resp. SuperSearcher$_{12}$).

where

$$\mathsf{constant} = \begin{cases} 20 & \text{if } d = 2, \\ 17.45 & \text{if } d = 4, \\ 27.68 & \text{if } d = 12. \end{cases}$$

5 RadicalSearcher$_\ell$: Efficient Subfield Curve Searching through radical ℓ-isogenies

One of the main disadvantages of the algorithm SuperSolver is its incompatibility with radical isogeny formulas. For example, the formulas from [5,8,4] work over the curve coefficients instead of the j-invariants, which (naively) would increase the cost per step from one square root into one field radical exponentiation along one field inversion. As highlighted in [23], those operations overkill the savings from employing the algorithm SuperSolver. In particular, such radical isogeny improvement was left as an open problem in [23].

Our proposed subfield root detection algorithm ScaledNeighbourInFp allows the compatibility with radical isogeny formulas [5,8,4] to find subfield curves in the general case (i.e., with slightly more generic primes).

To illustrate, let us focus on the radical 3-isogeny scenario. Therefore, we assume that $p = 4 \cdot f - 1$ for some positive integer f. Additionally, we work on the supersingular curve model $\mathcal{E}\colon y^2+a_1xy+a_3y=x^3$ for some $a_1,a_3\in\mathbb{F}_{p^2}$. We summarize the subfield searching algorithm in Figure 6.

```
RadicalSearcher_3($\mathcal{F}, h, L$)
1 :  Initialize an empty stack for storage of nodes, S
2 :  Deterministically find a curve $\mathcal{E}\colon y^2+a_1xy+a_3y=x^3$ isomorphic to $\mathcal{F}$
3 :  Push the root $\mathbf{z}_\varnothing=\{(a_1,a_3)),\varnothing\}$ onto S
4 :  Pop the first element in S, $\mathbf{z}_\mathsf{b}=\{(a_{1,\mathsf{b}},a_{3,\mathsf{b}}),\mathsf{b}\}$ where $\mathsf{b}\in\{0,1,2\}^n$
5 :  Compute the den. $u$ and num. $v$ of $j(\mathcal{E}_\mathsf{b})$ where $\mathcal{E}_\mathsf{b}\colon y^2+a_{1,\mathsf{b}}xy+a_{3,\mathsf{b}}y=x^3$
6 :  if $j(\mathcal{E}_\mathsf{b})\in\mathbb{F}_p$
7 :    return $(\mathsf{b},3)$
8 :  endif
9 :  for for each $\ell\in L$
10 :   if ScaledNeighbourInFp($\ell,u,v$)
11 :     return $(\mathsf{b},\ell)$
12 :   endif
13 : endfor
14 : if $n\leq(h-1)$
15 :   Calculate $\alpha=\sqrt[3]{-a_{3,\mathsf{b}}}$
16 :   for $k:=0$ to $2$
17 :     $a_{1,\mathsf{b}k}\leftarrow-6\alpha+a_{1,\mathsf{b}}$
18 :     $a_{3,\mathsf{b}k}\leftarrow 3a_{1,\mathsf{b}}\alpha^2-(a_{1,\mathsf{b}})^2\alpha+9a_{3,\mathsf{b}}$
19 :     Update $\alpha\leftarrow\zeta_3\alpha$
20 :   endfor
21 :   Push $\mathbf{z}_{\mathsf{b}2}=\{\mathcal{E}_{\mathsf{b}2},\mathsf{b}2\}$ onto S
22 :   Push $\mathbf{z}_{\mathsf{b}1}=\{\mathcal{E}_{\mathsf{b}1},\mathsf{b}1\}$ onto S
23 :   Push $\mathbf{z}_{\mathsf{b}0}=\{\mathcal{E}_{\mathsf{b}0},\mathsf{b}0\}$ onto S
24 :   Go to Line 4
25 : endif
26 : if S is empty
27 :   return $\perp$
28 : else
29 :   Go to Line 4
30 : endif
```

Fig. 6. Description of the optimized subfield searching algorithm using radical 3-isogenies. In practice, L is a list of small (odd prime) numbers and $3\notin L$. The integer n determines the length of the string b, and $\zeta_3\in\mathbb{F}_{p^2}$ is a cube-root of unity. For the empty bitstring $\mathsf{b}=\varnothing$, we set $n=0$. The check $j(\mathcal{E}_\mathsf{b})$ can be done using u and v without a field inversion.

A concrete running time for $\mathsf{RadicalSearcher}_3$: Let us focus on the scenario when $p = 4f - 1$ with $f = 3g$ and $\gcd(3, g) = 1$. Therefore, we have

$$\zeta_3 = \frac{-1 + i\sqrt{3}}{2} \quad \text{and} \quad \sqrt[3]{a} = a^{N_3} \quad \text{with} \quad N_3 = 3^{-1} \bmod \left(\frac{1}{3}(p^2 - 1)\right).$$

Notice that for concrete cryptographic parameters, we have that N_3 has a Hamming weight equal to $\frac{1}{2}\log_2 N_3 = (\log_2 p - 2)$. Consequently, computing a cube root requires

$$\begin{aligned}(\log_2 p - 2)\mathsf{M} + 2(\log_2 p - 2)\mathsf{S} &= (\log_2 p - 2)(3\mathsf{mul}) + 2(\log_2 p - 2)(2\mathsf{mul}) \\ &= 7(\log_2 p - 2)\mathsf{mul}.\end{aligned}$$

Now, apart from the cube root calculations per step in $\mathsf{RadicalSearcher}_3$, we need to compute $9\mathsf{M} + 6\mathsf{S} = 39\mathsf{mul}$ operations. Consequently, the algorithm $\mathsf{RadicalSearcher}_3$ performs

$$\mathsf{cost}_\mathsf{s} = 7(\log_2 p - 2)\mathsf{mul} + 39\mathsf{mul} = (7\log_2 p + 25)\mathsf{mul}$$

operations per step. On the other hand, the j-invariant of $\mathcal{E}_\mathsf{b}$ coincides with

$$j(\mathcal{E}_\mathsf{b}) = \frac{(a_{1,\mathsf{b}})^3 \cdot \left((a_{1,\mathsf{b}})^3 - 24a_{3,\mathsf{b}}\right)^3}{(a_{3,\mathsf{b}})^3 \cdot \left((a_{1,\mathsf{b}})^3 - 27a_{3,\mathsf{b}}\right)}.$$

Therefore, we can determine if $j(\mathcal{E}_\mathsf{b}) \in \mathbb{F}_p$ at the cost of $5\mathsf{M}+3\mathsf{S}+2\mathsf{mul} = 23\mathsf{mul}$ operations.

Hence, we consider the running time of algorithm $\mathsf{RadicalSearcher}_3$ to be bounded by

$$\mathsf{runtime}_3 = \underbrace{\frac{1}{3^{\left\lfloor \log_3\left(1+\sum_{\ell\in L} d_\ell\right)\right\rceil}}\left(\frac{1}{2}\mathsf{cost}_\mathsf{s} + \frac{3}{2}\Big(23 + \sum_{\ell \in L} \overbrace{d_\ell \cdot \mathsf{cost}_\ell^*}^{\mathsf{ScaledNeighbourInFp}}\Big)\right)}_{\gamma_p(L)} \cdot 3^h \; \mathsf{mul}.$$

Similarly as in Section 3, we focus on minimizing the value of $\frac{1}{\log_2 p} \cdot \gamma_p(L) \leq \frac{\mathsf{cost}_\mathsf{eqn}}{\log_2 p} \in O(1)$. In particular, we take the concrete costs of cost_ℓ from Table 4 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

$$\operatorname*{argmin}_{\mathbf{b}} \frac{1}{\log_2 p} \cdot \gamma_p(L_\mathbf{b}),$$

where $L_\mathbf{b}$ denotes the list having the i^{th} odd prime ℓ_i if the i^{th} bit of $\mathbf{b}$ is equal to one. Our experiments illustrate that the running time of $\mathsf{RadicalSearcher}_3$ is bounded by

$$\mathsf{runtime}_3 = c \cdot \log_2 p \cdot 3^h \ \mathsf{mul} = c \cdot \log_2 p \cdot \left(\frac{\#\mathbb{S}_{p^2}}{\#\mathbb{S}_p} \right) \mathsf{mul} \tag{11}$$

where $0 < c \lesssim 1.44$.

On the expected average running time of $\mathsf{RadicalSearcher}_3$*:* Observe that the algorithm $\mathsf{RadicalSearcher}_3$ inspects

$$\frac{3}{2} \cdot 3^h \cdot (1 + \sum_{\ell \in L} d_\ell)$$

curves, where $h = \log_3 \left(\frac{\#\mathbb{S}_{p^2}}{\#\mathbb{S}_p} \right) - \lfloor \log_3 (1 + \sum_{\ell \in L} d_\ell) \rceil$. Hence, the expected *average* running time of the algorithm $\mathsf{RadicalSearcher}_3$ turns out to be

$$\frac{1}{2} \cdot \left(\frac{2 \cdot 3^{\lfloor \log_3 (1 + \sum_{\ell \in L} d_\ell) \rceil}}{3 \cdot (1 + \sum_{\ell \in L} d_\ell)} \cdot \mathsf{runtime}_3^* \right) = \left(\frac{3^{\lfloor \log_3 (1 + \sum_{\ell \in L} d_\ell) \rceil}}{3 \cdot (1 + \sum_{\ell \in L} d_\ell)} \cdot \mathsf{runtime}_3^* \right).$$

For example, for L_2 we have $0.43 \cdot \mathsf{runtime}_3^*$, while for L_{54} we obtain $0.57 \cdot \mathsf{runtime}_3^*$. In particular, when p increases, the constant c decreases slightly (see Table 5).

$\log_p$	32	64	96	128	192	256	384	512
c (max.)	1.44	0.84	0.57	0.44	0.31	0.24	0.18	0.14
c (avg.)	0.62	0.48	0.33	0.25	0.18	0.14	0.10	0.08
L	L_2	L_{54}	L_{54}	L_{54}	L_{54}	L_{54}	L_{54}	L_{54}

Table 5. The row corresponding to L presents the best choice of L. In particular, we have $L_2 = [5]$ and $L_{54} = [5, 7, 13, 17]$. The row corresponding with c (max.) determines the constant factor in the running time of $\mathsf{RadicalSearcher}_3$ (i.e., Equation (11)), while c (avg.) with the average running time of $\mathsf{RadicalSearcher}_3$.

6 Experimental Results

Since algorithms $\mathsf{Searcher}$ and $\mathsf{SuperSearcher}$ (resp. $\mathsf{RadicalSearcher}_3$) perform on top of 2-isogenies (resp. 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 $j_0 \xrightarrow{\phi_1} j_1 \xrightarrow{\phi_2} \cdots \xrightarrow{\phi_d} j_d$ 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 $d \in \{2, 4, 12\}$.

All our experiments rectify our theoretical results, highlighting the improvements against [23]. We provide a proof-of-concept implementation using SageMath [26], available at [21].

Experiments concerning Searcher *and* SuperSearcher*:* 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 [10] for primes with $a \approx \frac{1}{2}r$.

We include experiments for the algorithms Searcher and SuperSearcher for both cases $a \approx \frac{1}{2}r$ and $a \approx r$. 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. Additionally,

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

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

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

We also include experiments for the algorithms Searcher and SuperSearcher when $p = 2^a \cdot 3^b \cdot f - 1$.

Experiments concerning $\mathsf{RadicalSearcher}_3$*:* Additionally, we include some experiments for primes of the form $p = 4 \cdot 3 \cdot f - 1$ for some positive integer f such that $\gcd(3, f) = 1$. For simplicity, we assume f is a prime number and brute-force search the interval $[30, 40] \cap \mathbb{Z}$ for finding these kinds of primes. In all the experiments concerning $\mathsf{RadicalSearcher}_3$, we take as input $\mathcal{E}$ and $h = \frac{1}{2}\log_3 p - 1$.

On the experiments: Tables 6, 7, 9 and 10 present the average running time of solving 25 random instances using $\mathsf{Searcher}_d$, $\mathsf{SuperSearcher}_d$, $\mathsf{RadicalSearcher}_3$ and SuperSolver. We graphically illustrate the comparison between our proposed algorithms in Figures 7, 8, 10 and 11. The horizontal line in **black** ink determines the constant from Equation (6). 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.

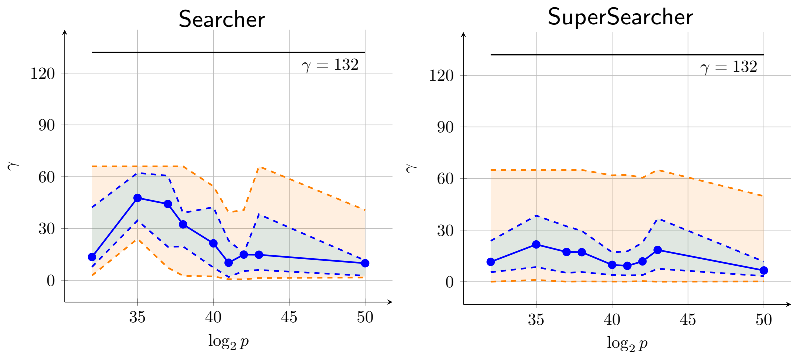
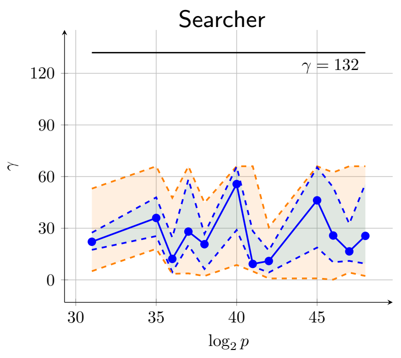


Fig. 7. These experiments correspond with the prime numbers $p = 2^a \cdot f - 1$ in Table 6.

Bits of p	$\{a, f\}$		Searcher	SuperSearcher
32	$\{17, 19417\}$	Running time: Success rate:	$23.7 \cdot 2^h$ 0.9	$15.8 \cdot 2^h$ 1.0
35	$\{21, 10645\}$	Running time: Success rate:	$47.5 \cdot 2^h$ 0.8	$26.4 \cdot 2^h$ 0.9
37	$\{19, 140995\}$	Running time: Success rate:	$40.6 \cdot 2^h$ 0.8	$20.9 \cdot 2^h$ 1.0
38	$\{19, 337825\}$	Running time: Success rate:	$31.5 \cdot 2^h$ 0.9	$20.3 \cdot 2^h$ 1.0
40	$\{22, 152945\}$	Running time: Success rate:	$25.0 \cdot 2^h$ 1.0	$11.8 \cdot 2^h$ 1.0
41	$\{21, 601135\}$	Running time: Success rate:	$14.0 \cdot 2^h$ 1.0	$13.1 \cdot 2^h$ 1.0
42	$\{22, 861533\}$	Running time: Success rate:	$13.5 \cdot 2^h$ 1.0	$16.4 \cdot 2^h$ 1.0
43	$\{22, 1924415\}$	Running time: Success rate:	$24.8 \cdot 2^h$ 0.8	$23.9 \cdot 2^h$ 0.9
50	$\{26, 9365135\}$	Running time: Success rate:	$10.9 \cdot 2^h$ 1.0	$8.7 \cdot 2^h$ 1.0

Table 6. All the numbers correspond with the average of solving 25 random instances by employing Searcher and SuperSearcher.

Some observations on the experiments for the case $p = 12^a \cdot f - 1$: Interestingly, our experiments concerning Tables 9 and 10 and Figures 10 and 11 highlight a lesser average running time than expected, about half than expected; such behavior seems to occur because of the form of the primes.



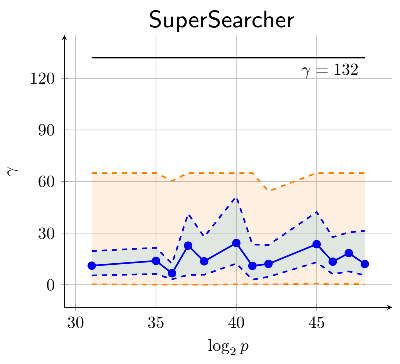
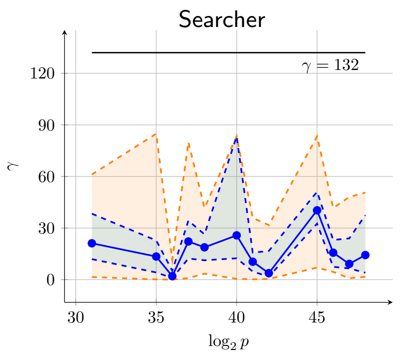


Fig. 8. These experiments correspond with the prime numbers $p = 2^a \cdot f - 1$ in Table 7.

Bits of p	$\{a, f\}$		Searcher	SuperSearcher
31	$\{31, 1\}$	Running time: Success rate:	$23.1 \cdot 2^h$ 1.0	$15.8 \cdot 2^h$ 1.0
35	$\{32, 5\}$	Running time: Success rate:	$39.1 \cdot 2^h$ 0.8	$17.9 \cdot 2^h$ 1.0
36	$\{34, 3\}$	Running time: Success rate:	$17.8 \cdot 2^h$ 1.0	$9.7 \cdot 2^h$ 1.0
37	$\{31, 43\}$	Running time: Success rate:	$36.3 \cdot 2^h$ 0.8	$26.9 \cdot 2^h$ 0.9
38	$\{33, 31\}$	Running time: Success rate:	$19.0 \cdot 2^h$ 1.0	$20.5 \cdot 2^h$ 0.9
40	$\{35, 31\}$	Running time: Success rate:	$46.6 \cdot 2^h$ 0.7	$30.6 \cdot 2^h$ 0.8
41	$\{36, 17\}$	Running time: Success rate:	$21.2 \cdot 2^h$ 0.9	$17.1 \cdot 2^h$ 1.0
42	$\{37, 31\}$	Running time: Success rate:	$11.7 \cdot 2^h$ 1.0	$15.7 \cdot 2^h$ 1.0
45	$\{39, 61\}$	Running time: Success rate:	$41.3 \cdot 2^h$ 0.7	$29.5 \cdot 2^h$ 0.8
46	$\{41, 19\}$	Running time: Success rate:	$29.9 \cdot 2^h$ 1.0	$18.7 \cdot 2^h$ 1.0
47	$\{40, 107\}$	Running time: Success rate:	$24.5 \cdot 2^h$ 0.9	$21.0 \cdot 2^h$ 1.0
48	$\{42, 53\}$	Running time: Success rate:	$31.1 \cdot 2^h$ 0.8	$19.6 \cdot 2^h$ 1.0

Table 7. Same description as in Table 6.



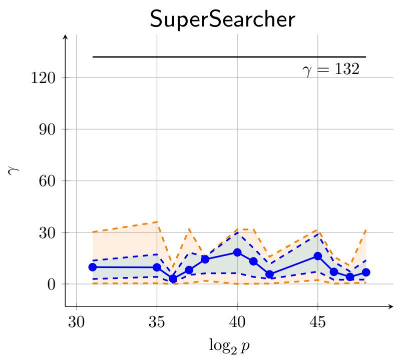


Fig. 9. These experiments correspond with the prime numbers $p = 2^a \cdot f - 1$ in Table 8.

Bits of p	$\{a, f\}$		Searcher$_4$	SuperSearcher$_4$
31	$\{31, 1\}$	Running time:	$24.6 \cdot 2^h$	$9.3 \cdot 2^h$
		Success rate:	1.0	1.0
35	$\{32, 5\}$	Running time:	$17.8 \cdot 2^h$	$11.5 \cdot 2^h$
		Success rate:	1.0	1.0
36	$\{34, 3\}$	Running time:	$3.4 \cdot 2^h$	$3.4 \cdot 2^h$
		Success rate:	1.0	1.0
37	$\{31, 43\}$	Running time:	$25.0 \cdot 2^h$	$12.1 \cdot 2^h$
		Success rate:	1.0	0.9
38	$\{33, 31\}$	Running time:	$19.7 \cdot 2^h$	$11.3 \cdot 2^h$
		Success rate:	0.9	0.6
40	$\{35, 31\}$	Running time:	$39.6 \cdot 2^h$	$17.9 \cdot 2^h$
		Success rate:	0.7	0.8
41	$\{36, 17\}$	Running time:	$10.8 \cdot 2^h$	$13.5 \cdot 2^h$
		Success rate:	1.0	0.9
42	$\{37, 31\}$	Running time:	$8.7 \cdot 2^h$	$6.9 \cdot 2^h$
		Success rate:	1.0	0.9
45	$\{39, 61\}$	Running time:	$40.4 \cdot 2^h$	$16.8 \cdot 2^h$
		Success rate:	0.9	0.7
46	$\{41, 19\}$	Running time:	$18.4 \cdot 2^h$	$7.7 \cdot 2^h$
		Success rate:	0.8	0.9
47	$\{40, 107\}$	Running time:	$16.1 \cdot 2^h$	$4.9 \cdot 2^h$
		Success rate:	1.0	1.0
48	$\{42, 53\}$	Running time:	$20.9 \cdot 2^h$	$9.4 \cdot 2^h$
		Success rate:	1.0	0.9

Table 8. Same description as in Table 6.

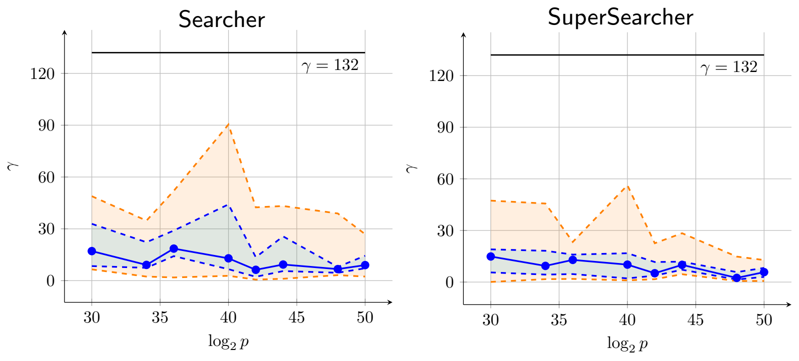


Fig. 10. These experiments correspond with the prime numbers $p = 12^a \cdot f - 1$ in Table 9.

Bits of p	$\{a, f\}$		Searcher$_{12}$	SuperSearcher$_{12}$
30	$\{4, 32783\}$	Running time: Success rate:	$22.5 \cdot 2^h$ 1.0	$16.0 \cdot 2^h$ 1.0
34	$\{5, 65599\}$	Running time: Success rate:	$14.3 \cdot 2^h$ 1.0	$14.2 \cdot 2^h$ 1.0
36	$\{5, 262151\}$	Running time: Success rate:	$21.5 \cdot 2^h$ 1.0	$11.3 \cdot 2^h$ 1.0
40	$\{6, 262231\}$	Running time: Success rate:	$27.4 \cdot 2^h$ 1.0	$13.5 \cdot 2^h$ 1.0
42	$\{6, 1048793\}$	Running time: Success rate:	$12.8 \cdot 2^h$ 1.0	$8.2 \cdot 2^h$ 1.0
44	$\{6, 4194493\}$	Running time: Success rate:	$16.2 \cdot 2^h$ 1.0	$11.3 \cdot 2^h$ 1.0
48	$\{7, 4194329\}$	Running time: Success rate:	$10.5 \cdot 2^h$ 1.0	$4.2 \cdot 2^h$ 1.0
50	$\{7, 16777499\}$	Running time: Success rate:	$11.6 \cdot 2^h$ 1.0	$6.0 \cdot 2^h$ 1.0

Table 9. Here, $p = 12^a \cdot f$ with f being a prime number. All the numbers concern the average of solving 25 random instances by employing Searcher$_{12}$ and SuperSearcher$_{12}$.

Therefore, it could be possible (in practice) to reduce the value of h to $\frac{1}{2}\log_2 p - 1$ (resp. $\frac{1}{2}\log_{12} p - 2$), and still being able to find a subfield curve when $p = 2^a \cdot 3^b \cdot f - 1$ with small f (resp. $p = 12^a \cdot f - 1$ with $a \approx \frac{1}{2}\log_{12} p$).

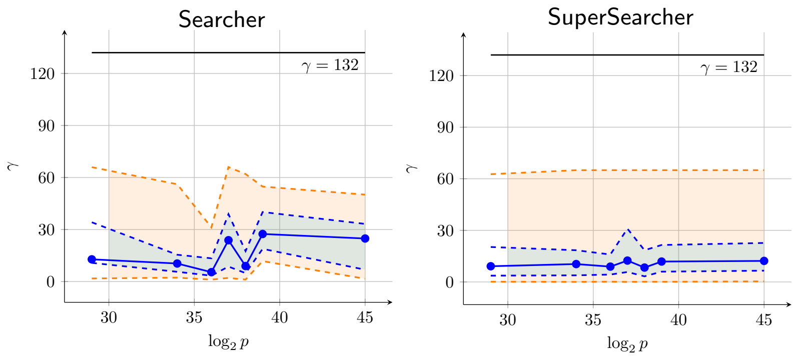


Fig. 11. These experiments concerns the prime numbers $p = 2^a \cdot f - 1$ in Table 10.

Bits of p	$\{a, f\}$		Searcher	SuperSearcher
29	$\{15, 15309\}$	Running time: Success rate:	$22.1 \cdot 2^h$ 0.9	$13.1 \cdot 2^h$ 1.0
34	$\{18, 45927\}$	Running time: Success rate:	$14.6 \cdot 2^h$ 1.0	$13.3 \cdot 2^h$ 1.0
36	$\{19, 98415\}$	Running time: Success rate:	$9.6 \cdot 2^h$ 1.0	$12.0 \cdot 2^h$ 1.0
37	$\{19, 216513\}$	Running time: Success rate:	$27.7 \cdot 2^h$ 0.9	$20.9 \cdot 2^h$ 0.9
38	$\{19, 413343\}$	Running time: Success rate:	$15.5 \cdot 2^h$ 1.0	$12.9 \cdot 2^h$ 1.0
39	$\{20, 413343\}$	Running time: Success rate:	$29.4 \cdot 2^h$ 1.0	$16.0 \cdot 2^h$ 1.0
45	$\{24, 1240029\}$	Running time: Success rate:	$22.7 \cdot 2^h$ 1.0	$17.5 \cdot 2^h$ 1.0

Table 10. Here $p = 2^a \cdot f$ and $f = 3^b \cdot g$ with $g \in \{5, 7, 11\}$ and $b \in \mathbb{Z}$. All the numbers concern the average of solving 25 random instances using Searcher and SuperSearcher.

7 Concluding Remarks

Our results in Section 4, backed up with our experimental results from Section 6, highlight a time complexity of

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

field multiplications and memory complexity polynomial in $\log_2 p$ for both algorithms Searcher$_d$ and SuperSearcher$_d$.

Bits of p	$\{a, f\}$		SuperSolver	RadicalSearcher$_3$
30	$\{2, 201326943\}$	Running time: Success rate:	$0.3 \cdot \log_2 p \cdot 2^h$ 1.0	$1.2 \cdot \log_2 p \cdot 2^h$ 0.9
32	$\{2, 805306611\}$	Running time: Success rate:	$0.4 \cdot \log_2 p \cdot 2^h$ 1.0	$1.7 \cdot \log_2 p \cdot 2^h$ 0.9
34	$\{2, 3221225817\}$	Running time: Success rate:	$0.5 \cdot \log_2 p \cdot 2^h$ 0.9	$1.9 \cdot \log_2 p \cdot 2^h$ 1.0
36	$\{2, 12884901933\}$	Running time: Success rate:	$0.2 \cdot \log_2 p \cdot 2^h$ 1.0	$0.5 \cdot \log_2 p \cdot 2^h$ 1.0
38	$\{2, 51539607627\}$	Running time: Success rate:	$0.5 \cdot \log_2 p \cdot 2^h$ 0.9	$0.9 \cdot \log_2 p \cdot 2^h$ 1.0
40	$\{2, 206158430733\}$	Running time: Success rate:	$0.2 \cdot \log_2 p \cdot 2^h$ 1.0	$0.7 \cdot \log_2 p \cdot 2^h$ 1.0

Table 11. All the numbers correspond with the average of solving 25 random instances by employing SuperSolver and RadicalSearcher$_3$.

Some remarks on the impact of our result on current constructions. We emphasize that our results should be taken as cautionary advice for proposing 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 $\lambda \in \{128, 192, 256\}$. Table 12 presents some concrete numbers concerning the impact of our results from Section 4.

Applicability to primes of the form $p = 2^a \cdot f + 1$. Our algorithms straightforwardly apply to those kinds of primes. The reasoning for that claim relies on the following two facts.

1. 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 $\mathbb{F}_{p^4}$ [10].
2. 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. However, 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 amount of multiplications and squares).

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 $(d - m)$.

	$\log_2 p$	**Delfs-Galbraith Algorithm**	**This work**
[2]	251	$2^{128.5}$	$2^{125.5}$
	383	$2^{194.6}$	$2^{191.5}$
	505	$2^{255.7}$	$2^{252.5}$
[14]	254	2^{130}	2^{127}
	413	$2^{209.6}$	$2^{206.5}$
	508	$2^{257.2}$	2^{254}
[19]	258	2^{132}	2^{129}
	386	2^{196}	2^{193}
	515	$2^{260.6}$	$2^{257.5}$
[12]	254	2^{130}	2^{127}
	389	$2^{197.6}$	$2^{194.5}$
	519	$2^{262.6}$	$2^{259.5}$
[24]	254	2^{130}	2^{127}

Table 12. On the impact of our result on current constructions. For the Delfs-Galbraith algorithm, we take the quantity $\log_2 p \cdot p^{1/2}$.

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 $\mathcal{E}[M]$.
- For the k^{th} 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 $\ell \approx M$. [8] 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 $\mathcal{O}(p^{1/2})$. [9]

[8] For the case of SuperSearcher and RadicalSearcher, one should assume ℓ is not divisible by some integer in the input list L.

[9] 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.

References

1. Adj, G., Cervantes-Vázquez, D., Chi-Domínguez, J.J., Menezes, A., Rodríguez-Henríquez, F.: On the cost of computing isogenies between supersingular elliptic curves. In: Cid, C., Jacobson, Jr., M.J. (eds.) SAC 2018. LNCS, vol. 11349, pp. 322–343. Springer, Cham (Aug 2019). https://doi.org/10.1007/978-3-030-10970-7_15
2. Basso, A., Dartois, P., De Feo, L., Leroux, A., Maino, L., Pope, G., Robert, D., Wesolowski, B.: SQIsign2D-west - the fast, the small, and the safer. In: Chung and Sasaki [9], pp. 339–370. https://doi.org/10.1007/978-981-96-0891-1_11
3. Bernstein, D.J., De Feo, L., Leroux, A., Smith, B.: Faster computation of isogenies of large prime degree. ANTS XIV–Proceedings of the Fourteenth Algorithmic Number Theory Symposium **4**(1), 39–55 (2020). https://doi.org/10.2140/obs.2020.4.39
4. Castryck, W., Decru, T., Houben, M., Vercauteren, F.: Horizontal racewalking using radical isogenies. In: Agrawal, S., Lin, D. (eds.) ASIACRYPT 2022, Part II. LNCS, vol. 13792, pp. 67–96. Springer, Cham (Dec 2022). https://doi.org/10.1007/978-3-031-22966-4_3
5. Castryck, W., Decru, T., Vercauteren, F.: Radical isogenies. In: Moriai and Wang [18], pp. 493–519. https://doi.org/10.1007/978-3-030-64834-3_17
6. Charles, D.X., Lauter, K.E., Goren, E.Z.: Cryptographic hash functions from expander graphs. Journal of Cryptology **22**(1), 93–113 (Jan 2009). https://doi.org/10.1007/s00145-007-9002-x
7. Chavez-Saab, J., Santos, M.C.R., De Feo, L., Eriksen, J.K., Hess, B., Kohel, D., Leroux, A., Longa, P., Meyer, M., Panny, L., Patranabis, S., Petit, C., Rodríguez-Henríquez, F., Schaeffler, S., Wesolowski, B.: SQIsign. Tech. rep., National Institute of Standards and Technology (2023), available at https://csrc.nist.gov/Projects/pqc-dig-sig/round-1-additional-signatures
8. Chi-Domínguez, J.J., Reijnders, K.: Fully projective radical isogenies in constant-time. In: Galbraith, S.D. (ed.) CT-RSA 2022. LNCS, vol. 13161, pp. 73–95. Springer, Cham (Mar 2022). https://doi.org/10.1007/978-3-030-95312-6_4
9. Chung, K.M., Sasaki, Y. (eds.): ASIACRYPT 2024, Part III, LNCS, vol. 15486. Springer, Singapore (Dec 2024)
10. Costello, C.: B-SIDH: Supersingular isogeny Diffie-Hellman using twisted torsion. In: Moriai and Wang [18], pp. 440–463. https://doi.org/10.1007/978-3-030-64834-3_15
11. Costello, C., Hisil, H.: A simple and compact algorithm for SIDH with arbitrary degree isogenies. In: Takagi, T., Peyrin, T. (eds.) ASIACRYPT 2017, Part II. LNCS, vol. 10625, pp. 303–329. Springer, Cham (Dec 2017). https://doi.org/10.1007/978-3-319-70697-9_11
12. Dartois, P., Leroux, A., Robert, D., Wesolowski, B.: SQIsignHD: New dimensions in cryptography. In: Joye and Leander [15], pp. 3–32. https://doi.org/10.1007/978-3-031-58716-0_1
13. Delfs, C., Galbraith, S.D.: Computing isogenies between supersingular elliptic curves over $\mathbb{F}_p$. DCC **78**(2), 425–440 (2016). https://doi.org/10.1007/s10623-014-0010-1
14. Duparc, M., Fouotsa, T.B.: SQIPrime: A dimension 2 variant of SQISignHD with non-smooth challenge isogenies. In: Chung and Sasaki [9], pp. 396–429. https://doi.org/10.1007/978-981-96-0891-1_13
15. Joye, M., Leander, G. (eds.): EUROCRYPT 2024, Part I, LNCS, vol. 14651. Springer, Cham (May 2024)

16. Kohel, D.: 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
17. Montgomery, P.L.: Speeding the pollard and elliptic curve methods of factorization. Mathematics of computation **48**(177), 243–264 (1987)
18. Moriai, S., Wang, H. (eds.): ASIACRYPT 2020, Part II, LNCS, vol. 12492. Springer, Cham (Dec 2020)
19. Nakagawa, K., Onuki, H., Castryck, W., Chen, M., Invernizzi, R., Lorenzon, G., Vercauteren, F.: SQIsign2D-east: A new signature scheme using 2-dimensional isogenies. In: Chung and Sasaki [9], pp. 272–303. https://doi.org/10.1007/978-981-96-0891-1_9
20. Onuki, H., Nakagawa, K.: Ideal-to-isogeny algorithm using 2-dimensional isogenies and its application to SQIsign. In: Chung and Sasaki [9], pp. 243–271. https://doi.org/10.1007/978-981-96-0891-1_8
21. Public Repository: The code is going to be public soon
22. Renes, J.: Computing isogenies between Montgomery curves using the action of (0, 0). In: Lange, T., Steinwandt, R. (eds.) Post-Quantum Cryptography - 9th International Conference, PQCrypto 2018. pp. 229–247. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-79063-3_11
23. Santos, M.C.R., Costello, C., Shi, J.: Accelerating the delfs-galbraith algorithm with fast subfield root detection. In: Dodis, Y., Shrimpton, T. (eds.) CRYPTO 2022, Part III. LNCS, vol. 13509, pp. 285–314. Springer, Cham (Aug 2022). https://doi.org/10.1007/978-3-031-15982-4_10
24. Santos, M.C.R., Eriksen, J.K., Meyer, M., Reijnders, K.: AprèsSQI: Extra fast verification for SQIsign using extension-field signing. In: Joye and Leander [15], pp. 63–93. https://doi.org/10.1007/978-3-031-58716-0_3
25. Scott, M.: A note on the calculation of some functions in finite fields: Tricks of the trade. Cryptology ePrint Archive, Report 2020/1497 (2020), https://eprint.iacr.org/2020/1497
26. The Sage Developers: SageMath, the Sage Mathematics Software System (Version 10.1) (2023), `https://www.sagemath.org`