Let us walk on the 3-isogeny graph: efficient, fast, and simple

Jesús-Javier Chi-Domínguez, Eduardo Ochoa-Jiménez and Ricardo-Neftalí Pontaza-Rodas

Technology Innovation Institute, Abu Dhabi, UAE
{jesus.dominguez,eduardo.ochoa,ricardo.pontaza}@tii.ae

Abstract. Constructing and implementing isogeny-based cryptographic primitives is an active research. In particular, performing length-n isogenies walks over quadratic field extensions of $\mathbb{F}_p$ plays an exciting role in some constructions, including Hash functions, Verifiable Delay Functions, Key-Encapsulation Mechanisms, and generic proof systems for isogeny knowledge. Remarkably, many isogeny-based constructions, for efficiency, perform 2-isogenies through square root calculations.
This work analyzes the idea of using 3-isogenies instead of 2-isogenies, which replaces the requirement of calculating square roots with cube roots. Performing length-m 3-isogenies allows shorter isogeny walks than when employing length-n 2-isogenies since a cube root calculation costs essentially the same as computing a square root, and we require $3^m \approx 2^n$ to provide the same security level.
We propose an efficient mapping from arbitrary supersingular Montgomery curves defined over $\mathbb{F}_{p^2}$ to the 3-isogeny curve model from Castryck, Decru, and Vercauteren (Asiacrypt 2020); a deterministic algorithm to compute all order-3 points on arbitrary supersingular Montgomery curves, and an efficient algorithm to compute length-m 3-isogeny chains.
We improve the length-m 3-isogeny walks required by the KEM from Nakagawa and Onuki (CRYPTO 2024) by using our results and introducing more suitable parameter sets that are friendly with C-code implementations. In particular, our experiments illustrate an improvement of 26.41%–35.60% in savings when calculating length-m 3-isogeny chains and using our proposed parameters instead of those proposed by Nakagawa and Onuki (CRYPTO 2024).
Finally, we enhance the key generation of CTIDH-2048 by including radical 3-isogeny chains over the basefield $\mathbb{F}_p$, reducing the overhead of finding a 3-torsion basis as required in some instantiations of the CSIDH protocol. Our experiments illustrate the advantage of radical 3 isogenies in the key generation of CTIDH-2048, with an improvement close to 4x faster than the original dCTIDH.

Keywords: CGL-hash function · CTIDH · Isogeny-based cryptography · Post-quantum cryptography · Radical 3-isogenies

1 Introduction

In 2009, Charles, Gordon, and Lauter [CLG09] proposed a hash function, namely CGLHash, based on the computation of isogenies (of a large degree) between supersingular elliptic curves over finite fields, $\mathbb{F}_{p^2}$. Since the introduction of the CGLHash function, supersingular isogeny problems have attracted considerable attention in cryptography, because the best-known attacks have exponential complexity. Nevertheless, the main drawback of the CGLHash function is its efficiency, which requires $\log_2(p)$ field multiplications per input bit. In practice, the CGLHash function is instantiated for efficiency via 2-isogenies (requiring a single square root per bit), which summarizes as calculating length-n 2-isogeny walks (i.e., involves the calculation of n 2-isogenies).

Remark 1. Calculating length-n 2-isogenies walks via square roots is essential in cryptanalysis; the best classical attack [DG16, SCS22] requires a single square root per 2-isogeny.

Related work: computing isogenies over $\mathbb{F}_p$ via radical calculations. In 2020, Castryck and Decru [CD20] presented an algorithm for calculating length-n 2-isogeny walks formulas between supersingular elliptic curves defined over $\mathbb{F}_p$. Their algorithm performs 2-isogenies via square roots over $\mathbb{F}_p$, and was dedicated to improve the isogeny-based group action from [CLM+18]. Subsequently, Castryck, Decru and Vercauteren extended the work from [CD20] by calculating ℓ-isogenies through radical calculations (i.e., calculating ℓ-th roots). Several consecutive works [OM22, CR22, CDHV22, Dec24] have recently extended and improved the radical formulas from [CDV20] for $\ell > 3$. The current most efficient variant of the isogeny-based Diffie-Hellman protocol from [CLM+18] requires calculating 3-isogenies [BBC+21, CCSC+24, CHMR25].

Related work: computing isogenies over $\mathbb{F}_{p^2}$ via radical calculations. In 2022, Chávez-Saab, Rodríguez-Henríquez, and Tibouchi [CSRT22] described an isogeny-based Verifiable Delay Function (VDF) having as core the calculation of length-n 2-isogeny walks, as required in the CGLHash function, needing a single square root per 2-isogeny. In 2023, Cong, Lai and Levin [CLL23] detailed a non-interactive protocol to prove the knowledge of an isogeny path using a generic zkSNARK proof system, which performs 2-isogenies via square roots. Recently, Nakagawa and Onuki [NO24] presented a Key Encapsulation Mechanism (KEM) that involves calculating length-m 3-isogeny walks, where each 3-isogeny requires a cube root computation. In addition, Levin and Pedersen [LP24] described a Verifiable Random Function (VRF) based on 2-isogenies calculated through square roots.

Our Contribution: In this work, we are interested in improving the calculation of isogeny walks such as those used in [NO24, BBC+21, CCSC+24] through the usage of radical 3-isogenies. In general, our results could help to improve any isogeny-based construction where the chain of radical 3-isogenies is required instead of 2-isogenies or finding for order-3 points are needed. The core contributions of this paper are the following.

1. We present an efficient mapping from Montgomery curve models to the 3-isogeny curve model from [CDV20]. More precisely, given a Montgomery curve $\mathcal{F}\colon y^2 = x^3+Ax^2+x$ and the x-coordinate x_P of an order-3 point P on $\mathcal{F}$, we show that $\mathcal{F}$ is isomorphic to $\mathcal{G}\colon y^2+a_1xy+a_3y = x^3$ with $a_1 = 3x_P^2+2Ax_P+1$ and $a_3 = 2(x_P^3 + Ax_P + x_P)^2$, requiring only two multiplications, two squares, and six additions in $\mathbb{F}_{p^2}$.

2. We describe a deterministic and efficient procedure for computing all the order-3 points on arbitrary supersingular Montgomery curves. This procedure relies on solving the quartic equation determined by the division-3 polynomial of the Montgomery curve $\mathcal{F}$. We illustrate the advantage of our proposal in Table 1.

Table 1: Comparison between calculating (x-coordinates of) order-3 points on an arbitrary supersingular Montgomery curve $\mathcal{F}$. Here, mul_p denotes a field multiplication in $\mathbb{F}_p$.

	Approach	Type of algorithm	#(Points computed)	Cost
[CDV20, OM22]	Sampling an order-3 point	Probabilistic	One [1]	$50\log_2(p)\mathsf{mul}_p$
This work	Solving a quartic equation	Deterministic	Four	$14.5\log_2(p)\mathsf{mul}_p$

3. We explicitly and efficiently describe how to compute length-m 3-isogeny chains with non-backtracking via the 3-isogeny formulas from [CDV20]. In particular, we show that using the 3-isogeny curve model $\mathcal{E}\colon y^2 + a_1xy + a_3y = x^3 - 5a_1a_3x - a_1^3a_3 - 7a_3^2$ allows to compute length-m 3-isogeny chains with non-backtracking.

Our analysis centers on supersingular elliptic curves defined over $\mathbb{F}_{p^2}$ with $p \equiv 3 \bmod 4$ and $p^2 - 1 = 3\cdot f$ for some integer $f \in \mathbb{N}$ such that f is relatively prime to three. [2] We verify the correctness of our results through intensive experiments and illustrate the advantages and differences of performing 3-isogeny walks instead of 2-isogeny walks (see Table 2). Our experiments show that the $\mathsf{CGLHash}_3$ function has a competitive performance compared to the $\mathsf{CGLHash}_2$ function.

Table 2: Comparison between computing 2-isogeny and 3-isogeny walks with $p \equiv 3 \bmod 4$.

	2-isogeny walks [CLG09]	3-isogeny walks (this work)
Condition on p	—	$p^2 - 1 = 3 \cdot f$ with $\gcd(3, f) = 1$.
Initial setup	The j-invariants $j(\mathcal{E}_{-1})$ and $j(\mathcal{E}_0)$ of two fixed 2-isogenous curves.	An order-3 point P on $\mathcal{F}_{-1}\colon y^2 = x^3 + Ax^2 + x$ along with $\mathcal{F}_{-1}$.
Path nodes	$j(\mathcal{E}_i)$	$\mathcal{E}_i\colon y^2 + a_1xy + a_3y = x^3 - 5a_1a_3x - a_1^3a_3 - 7a_3^2$
Path length	n	$n/\log_2(3)$
Bottleneck operation	Square root calculation	Cube root calculation
Additional calculations	—	The j-invariant $j(\mathcal{E}_n)$ of $\mathcal{E}_n$

Our contribution on the efficient method for computing order-3 points (the second contribution from above) has the following two important implications.

First, it allows us to provide some less conservative parameter sets for instantiating [NO24], which still provide the desired bit security according to the complexity $\tilde{O}(p^{1/2})$. More precisely, we propose three primes (named $p381$, $p575$ and $p765$) to reach levels equal to QFESTA-128, QFESTA-192 and QFESTA-256, and compare them against the primes proposed by [NO24] (namely, $p398$, $p592$, and $p783$). Our suggested parameters benefit from having one less 64-bit word, giving a performance improvement of 35,60% for 128-bits, 31.62% for 192-bits, and 26.41% for 256-bits for CPU cycles.

[1] This approach gets a random order-3 point with probability 0.96. We detail the analysis discussion in Section 3.2.

[2] Our results holds for any prime, but for efficiency and simplicity we limit our analysis to $p^2 - 1 = 3\cdot f$ with $\gcd(3, f) = 1$. We only require an efficient field exponentiation for computing cube-roots in $\mathbb{F}_{p^2}$.

Second, it allows to reduce the overhead for the initial step of chains of radical 3-isogenies (finding order-3 points over $\mathbb{F}_p$ and mapping to the correct curve model). In particular, we provide projective formulas instantiation and implementation of [OM22]. Using our results we improve the key generation of the CTIDH protocol (e.g., dCTIDH [BBC+21, CHMR25]). Our experiments illustrate a speed-up of 4x faster compared to the original implementation of dCTIDH.

As part of our results, we provide the first x86_64 Intel C-code implementation for computing 3-isogeny walks on arbitrary supersingular Montgomery curves defined over $\mathbb{F}_{p^2}$ via cube-root calculations and present an efficient implementation of the $\mathsf{CGLHash}_3$ function. We use a custom assembly code designed for x86_64 Intel to guarantee that our implementation is constant-time compliant. Additionally, our implementation is memory-leak free, aiming for a stronger security solution. In particular, we provide two dedicated implementations [3]:

1. An implementation that performs 3-isogeny random walk over $\mathbb{F}_{p^2}$ (as required in [NO24]). Figure 1 describes our results and the operation blocks needed to implement 3-isogeny walks over $\mathbb{F}_{p^2}$.

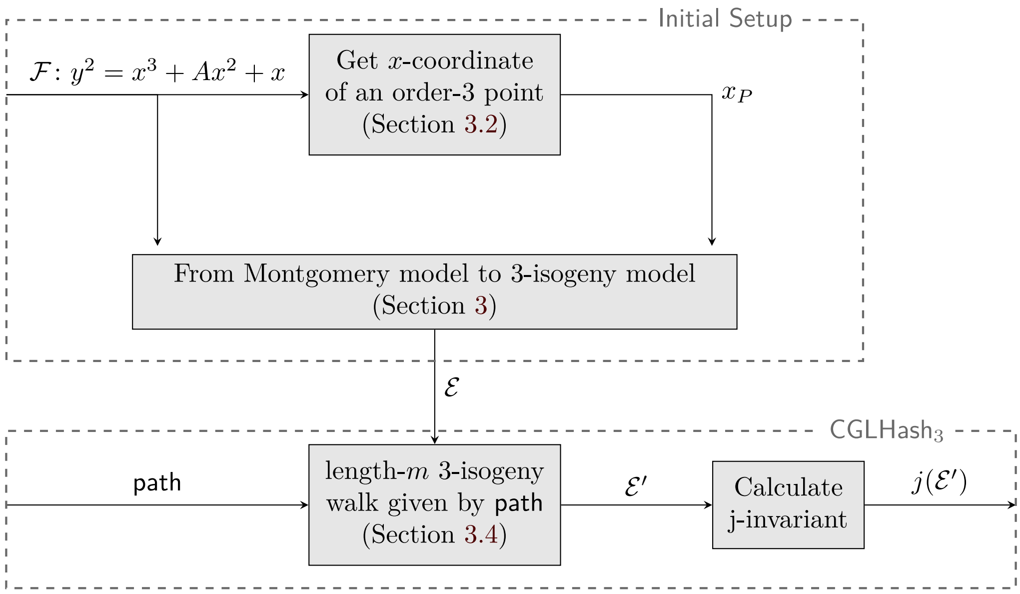


Figure 1: The superingular elliptic curves $\mathcal{E}$ and $\mathcal{E}'$ are defined over $\mathbb{F}_{p^2}$ and described as in Section 3.5.

2. An implementation that calculates chains of 3-isogenies over $\mathbb{F}_p$, with the goal of improving [BBC+21, CHMR25]. Figure 2 describes our results and the operation blocks needed to implement 3-isogeny chains over $\mathbb{F}_p$.

Outline. Section 2 presents the necessary mathematical background. Section 3 presents our theoretical results. In particular, Section 3.2 describes a deterministic method for calculating order-3 points over arbitrary supersingular Montgomery curves defined over $\mathbb{F}_{p^2}$, while Sections 3.3 and 3.4 detail how to perform length-m 3-isogeny walks with non-backtracking. Subsequently, Section 3.5 provides an optimized description of the CGLHash function instantiated with 3-isogenies, and Section 3.6 analyzes the impact of our results

[3] Our C-code implementation will be public soon.

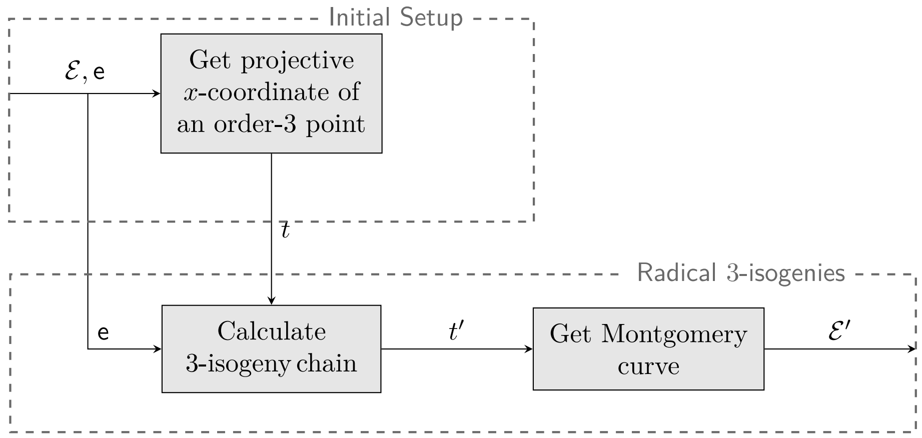


Figure 2: The superingular elliptic curves $\mathcal{E}$ and $\mathcal{E}'$ are defined over $\mathbb{F}_p$ and given in Montgomery curve form. All the block calculations are presented in Section 3.7.

on the KEM construction of [NO24]. Section 3.7 shows how our results can be used to improve the CTIDH protocol. We finally present our experiments in Section 4. Section 4.1 includes experiments over $\mathbb{F}_{p^2}$, while Section 4.2 over $\mathbb{F}_p$.

2 Preliminaries

Let p be a prime integer number with $p \equiv 3 \bmod 4$. We focus on the quadratic field extension $\mathbb{F}_{p^2} = \mathbb{F}_p[i]/(i^2+1)$ of $\mathbb{F}_p$ and supersingular elliptic curves $\mathcal{E}$ with $\#\mathcal{E}(\mathbb{F}_{p^2}) = (p \pm 1)^2$ points.

Unless we specify a different model, we center on elliptic curves in Montgomery form (i.e, $\mathcal{E}\colon y^2 = x^3 + Ax^2 + x$ for some $A \in \mathbb{F}_{p^2}$). Elliptic curves are uniquely determined (up to isomorphism) by their j-invariant, which in this case coincides with $j(\mathcal{E}) = 256(A^2-3)^3/(A^2-4)$.

An isogeny $\varphi\colon \mathcal{E} \to \mathcal{F}$ is a non-constant morphism between two elliptic curves that has a finite kernel and sends the point at infinity $\mathbf{0}_{\mathcal{E}}$ on $\mathcal{E}$ to the point at infinity $\mathbf{0}_{\mathcal{F}}$ on $\mathcal{F}$ (i.e., $\varphi(\mathbf{0}_{\mathcal{E}}) = \mathbf{0}_{\mathcal{F}}$). In particular, we focus on cyclic isogenies (i.e., $\ker\varphi$ is cyclic), and we say that φ is an ℓ-isogeny if $\ker\varphi$ has size ℓ. The dual of φ is another ℓ-isogeny $\hat{\varphi}\colon \mathcal{F} \to \mathcal{E}$ such that $\varphi \circ \hat{\varphi} = [\ell]$ and $\hat{\varphi} \circ \varphi = [\ell]$, 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 $\hat{\varphi}\colon \mathcal{F} \to \mathcal{E}$.

Division and Modular polynomials. The ℓ-division polynomial $\Psi_{\mathcal{E},\ell}(x)$ of an elliptic curve $\mathcal{E}$ determines all the ℓ-torsion points on $\mathcal{E}$ (i.e., all the points P on $\mathcal{E}$ such that $[\ell]P = \mathbf{0}_{\mathcal{E}}$); more precisely, its roots determine the x-coordinates of all the ℓ-torsion points on $\mathcal{E}$. The ℓ^{th} modular polynomial $\Phi_\ell(x,y) \in \mathbb{Z}[x,y]$ is an irreducible polynomial such that $\Phi_\ell(j(\mathcal{E}), j(\mathcal{F})) = 0$ for any pair of two ℓ-isogenous curves $\mathcal{E}$ and $\mathcal{F}$.

Charles-Goren-Lauter hash function. Given two ℓ-isogenous supersingular elliptic curves $\mathcal{E}_0$ and $\mathcal{E}_{-1}$, and an length-n ℓ-isogeny chain

$$\mathcal{E}_0 \xrightarrow{\varphi_1} \mathcal{E}_1 \xrightarrow{\varphi_2} \cdots \xrightarrow{\varphi_n} \mathcal{E}_n \tag{1}$$

with non-backtracking (i.e., φ_{i+1} different from $\hat{\varphi}_i$), the ℓ-isogeny path connecting $\mathcal{E}_0$ and $\mathcal{E}_n$ can be encoded by a length-n list of integers in $[1, \ell-1]$. The Charles-Goren-Lauter hash function [CLG09] determined by $\mathcal{E}_0$ and $\mathcal{E}_{-1}$ is defined as

$$\begin{aligned}\mathsf{CGLHash}_\ell \colon \{1,\ldots,\ell-1\}^* &\to \mathbb{F}_{p^2} \\ \mathsf{path} &\mapsto j(\mathcal{E}_n)\end{aligned} \tag{2}$$

where $n = \#\mathsf{path}$ and $\mathcal{E}_n$ is the end curve of the length-n ℓ-isogeny chain determined by path. Commonly, ℓ is chosen as a small prime number (e.g., $\ell = 2$).

Computing 2-isogenies via square-roots. For any 2-isogeny $\varphi_i \colon \mathcal{E}_i \to \mathcal{E}_{i+1}$ in Equation (1), we have that $j_i \coloneqq j(\mathcal{E}_i)$ and $j_{i+1} \coloneqq j(\mathcal{E}_{i+1})$ annihilate the 2nd modular polynomial $\Phi_2(x, y)$. In particular, given the $j_{i-1} \coloneqq j(\mathcal{E}_{i-1})$ and j_i, the next j-invariant in Equation (1) can be calculated as below [CSRT22].

$$j_{i+1} = \frac{1}{2}\left(j_i^2 - 1288 j_i - j_{i-1} + 162000 \pm \sqrt{\rho_i}\right) \tag{3}$$

where

$$\begin{aligned}\rho_i = j_i^4 &- 2976 j_i^3 + 2 j_i^2 j_{i-1} + 2532192 j_i^2 - 2976 j_i j_{i-1} \\ &- 645205500 j_i - 3 j_{i-1}^2 + 324000 j_{i-1} - 8748000000.\end{aligned}$$

Assuming we have a deterministic algorithm for computing square-roots in $\mathbb{F}_{p^2}$, the i^{th} bit of an input $\mathsf{path} \in \{0,1\}^*$ for $\mathsf{CGLHash}_2$ determines the sign in $\pm\sqrt{\rho_i}$ in Equation (3). For example, one can deterministically calculate a square root over $\mathbb{F}_{p^2}$ using the Kong et al. algorithm [KCYL06] if $p^2 \equiv 9 \bmod 16$. However, Scott's algorithm [Sco20] allows computing square roots over $\mathbb{F}_{p^2} \setminus \mathbb{F}_p$ much more efficiently than the Kong et al. algorithm.

Computing 3-isogenies via cube-roots. There are two ways of computing radical 3-isogenies: Using the formulas from [CDV20], which are the best choice when we are working over $\mathbb{F}_{p^2}$ (e.g., [NO24]); or using the formulas from [OM22] when we are working over $\mathbb{F}_p$ (e.g., [CLM+18]).

The authors in [CDV20, OM22] take an extra assumption, $p = 3 \cdot f - 1$ for some $f \in \mathbb{N}$ such that $4 \mid f$ and $\gcd(3, f) = 1$. Let N_3 be the inverse of three modulo $\frac{1}{3}(p^2 - 1)$ and $\zeta_3 = \frac{-1 + i\sqrt{3}}{2} \in \mathbb{F}_{p^2}$ be a cube-root of unity. Let $a \in \mathbb{F}_{p^2}$ be a cubic residue, then $b = a^{N_3}$, $c = \zeta_3 b$, and $d = \zeta_3^2 b$ are the three cube-roots of a.

Let $\mathcal{E} \colon y^2 + a_1 xy + a_3 y = x^3$ be a supersingular elliptic curve defined over $\mathbb{F}_{p^2}$, and let α be a cube-root of $-a_3$. Then the supersingular elliptic curve $\mathcal{E}' \colon y^2 + a_1' xy + a_3' y = x^3$ with $a_1' = (-6\alpha + a_1)$ and $a_3' = (3a_1\alpha^2 - a_1^2\alpha + 9a_3)$ is 3-isogenous to $\mathcal{E}$ [CDV20]. In particular, $\mathcal{E}'$ is isomorphic to the codomain of the 3-isogeny $\vartheta \colon \mathcal{E} \to \mathcal{E}''$ with kernel $\ker\vartheta = \langle (0,0) \rangle$ (i.e., $\mathcal{E}'$ is isomorphic to $\mathcal{E}''$).

Let $\mathcal{E} \colon y^2 = x^3 + Ax^2 + x$, and (t, y) be an order-3 point on $\mathcal{E}$ such that $(t, y) \in \mathcal{E}[\pi - \epsilon]$ with $\epsilon \in \{1, -1\}$. Then, $t' = 3t\alpha^2 + (3t^2 - 1)\alpha + 3t^3 - 2t$ determines the x-coordinate of an order-3 point $(t', y') \in \mathcal{E}'[\pi - \epsilon]$ where $\mathcal{E}' \colon y^2 = x^3 + A'x^2 + x$ is the codomain of the isogeny with kernel generated by (t, y) and $\alpha = \sqrt[3]{t(t^2 - 1)}$ [OM22]. In particular, it holds that $A' = \frac{-3(t')^4 - 6(t')^2 + 1}{4(t')^3}$.

Notation and baseline cost assumptions. Through this paper, we write I, M, S, and A to refer to inversions, multiplications, squares, and additions in $\mathbb{F}_{p^2}$. Similarly, inv_p, mul_p, sqr_p, and add_p denote inversions, multiplications, squares, and additions in $\mathbb{F}_p$. Additionally, we write E_2 and E_3 to refer to a square-root and cube-root in $\mathbb{F}_{p^2}$.

Remark 2. Determining if an element $a \in \mathbb{F}_p$ is a quadratic residue it reduces to verifying if $a^{\frac{p-1}{2}} = 1$. In addition, we have $\sqrt{a} = a^{\frac{p+1}{4}}$, whenever a has a square root over $\mathbb{F}_p$, and $\sqrt[3]{a} = a^{\frac{2p-1}{3}}$. Therefore, we write exp_p to refer to these calculations over $\mathbb{F}_p$. In particular, we assume $\mathsf{exp}_p = \mathsf{inv}_p$ in general.

Field inversions [4] and cube-roots correspond with raising to the powers of $(p-2)$ and $N_3 = 3^{-1} \bmod (\frac{1}{3}(p^2-1))$, which for concrete cryptographic parameters have approximately $\frac{1}{2}(p-2)$ and $\frac{1}{2}N_3$ as Hamming weights, respectively. On the other hand, Scott's algorithm requires raising elements in $\mathbb{F}_p$ to the power of $N_2 = \frac{1}{4}(p+1)$ and $M_2 = \frac{1}{4}(p-3)$. Therefore, for simplicity and without loss of generality, we assume that calculating field inversions, square roots, and cube roots takes $\log_2(p)(\mathsf{sqr}_p + \frac{1}{2}\mathsf{mul}_p) = \frac{2}{3}\log_2(p)\mathsf{M}$, $\log_2(p)(2\mathsf{sqr}_p + \mathsf{mul}_p)$, and $\log_2(p)(2\mathsf{S} + \mathsf{M})$ operations, respectively. [5]

3 On the efficiency of non-backtracking 3-isogeny walks

As highlighted in the introduction, some cryptographic applications explicitly require calculating long length-n chains of 3-isogenies

$$\mathcal{E}_0 \coloneqq \mathcal{E} \xrightarrow[\phi_1]{\text{3-isogeny}} \mathcal{E}_1 \xrightarrow[\phi_2]{\text{3-isogeny}} \mathcal{E}_2 \xrightarrow[\phi_3]{\text{3-isogeny}} \cdots \xrightarrow[\phi_n]{\text{3-isogeny}} \mathcal{E}' \coloneqq \mathcal{E}_n$$

(instead of 2-isogenies) over specific fields where no rational order-3^n kernel point exists; that is, where we cannot efficiently compute 3^n-isogenies through kernel point generators. On those cases, each 3-isogeny ϕ_j is then calculated through cube root calculations over the finite field $\mathbb{F}_{p^2}$ (e.g. [CDV20, OM22]), and, for efficiency, some additional requirements are needed. Below, we list the four main ingredients required for computing length-n 3-isogeny chains.

- As the first ingredient, we need an efficient algorithm for computing cube roots.
- We require an efficient mapping to switch to a suitable curve model. For example, the formulas from [CDV20] work over curves different from the Montgomery curve model.
- We required an initial order-3 point on the curve as the third ingredient. Therefore, we need an efficient algorithm for finding/calculating an order-3 point for arbitrary curves.
- Additionally, we must ensure that there is non-backtracking in the 3-isogeny chain.

In this section, we assume $p \equiv 3 \bmod 4$ and $(p^2-1) = 3 \cdot f$ for some positive integer $f \in \mathbb{N}$ such that $\gcd(3, f) = 1$. Therefore, we have an efficient procedure for computing cube roots (see Section 2).

3.1 An efficient transformation for getting the 3-isogeny curve model

Let $P = (x_P, y_P)$ be an order-3 point on a Montgomery curve $\mathcal{F}\colon y^2 = x^3 + Ax^2 + x$. The isomorphism $\iota\colon \mathcal{F} \to \mathcal{G}$ determined by the translation $P \mapsto (0,0)$ has codomain $\mathcal{G}\colon y^2 + a_1xy + a_3y = x^3$, where $a_1 = (3x_P^2 + 2Ax_P + 1)/y_P$ and $a_3 = 2y_P$. However, we can avoid calculating the inverse of y_P as follows.

Notice that j-invariant of $\mathcal{G}$ (in terms of a_1 and a_3) is equal to

$$j(\mathcal{G}) = \frac{a_1^3(a_1^3 - 24a_3)^3}{a_3^3(a_1^3 - 27a_3)}.$$

[4] A field inversion in $\mathbb{F}_{p^2}$ requires a single field inversion in $\mathbb{F}_p$ when $p \equiv 3 \bmod 4$.

[5] A multiplication (resp. square) in $\mathbb{F}_{p^2}$ requires three multiplications (resp. two multiplications) in $\mathbb{F}_p$ when $p \equiv 3 \bmod 4$.

Now, let us rewrite $a_1 = \tilde{a}_1/y_P$ with $\tilde{a}_1 = (3x_P^2 + 2Ax_P + 1)$, and set $\tilde{a}_3 = a_3 y_P^3 = 2y_P^4$. Observe that

$$j(\mathcal{G}) = \frac{a_1^3(a_1^3 - 24a_3)^3}{a_3^3(a_1^3 - 27a_3)} = \frac{\tilde{a}_1^3(\tilde{a}_1^3 - 24a_3 y_P^3)^3}{y_P^9 a_3^3(a_1^3 - 27a_3 y_P^3)} = \frac{\tilde{a}_1^3(\tilde{a}_1^3 - 24\tilde{a}_3)^3}{\tilde{a}_3^3(a_1^3 - 27\tilde{a}_3)} = j(\tilde{\mathcal{G}}),$$

where $\tilde{\mathcal{G}}\colon y^2 + \tilde{a}_1 xy + \tilde{a}_3 y = x^3$. Therefore, $\mathcal{G}$ and $\tilde{\mathcal{G}}$ are isomorphic over $\mathbb{F}_{p^2}$ and the isomorphism $\tilde{\iota}\colon \mathcal{G} \to \tilde{\mathcal{G}}$ is give by the map

$$\tilde{\iota}\colon (x, y) \mapsto (y_P^2 x, y/y_P^3).$$

It is important to stress that the order-3 point P either belongs to $\mathcal{F}[p+1]$ or $\mathcal{F}[p-1]$. Consequently, we have that x_P and y_P^2 lie in $\mathbb{F}_{p^2}$; thus, the curve $\tilde{\mathcal{G}}$ is defined over $\mathbb{F}_{p^2}$.

To sum up, given a Montgomery curve and an order-3 point, we can move to the 3-isogeny curve model from [CDV20] at the cost of $2\mathsf{M} + 2\mathsf{S} + 6\mathsf{A}$ operations.

3.2 Computing x-coordinates of order-3 points

Let us focus on a Montgomery curve $\mathcal{F}\colon y^2 = x^3 + Ax^2 + x$ with $A \in \mathbb{F}_{p^2}$. Notice that the 3-division polynomial of $\mathcal{F}$ is

$$\Psi_{\mathcal{F},3}(x) = 3x^4 + 4Ax^3 + 6x^2 - 1, \tag{4}$$

such that its roots determine the x-coordinates of order-3 points $P = (x_P, y_P)$ on $\mathcal{F}$. Observe that the construction of the 3-isogenous curve $\mathcal{G}\colon y^2 + (3x_P^2 + 2Ax_P + 1)xy + (2y_P^4)y = x^3$ can obtained from the pair (x_P, y_P^2); that is, y_P is not explicitly required. Thus, solving the quartic polynomial given by $\Psi_{\mathcal{F},3}(x)$ yields an efficient and deterministic procedure to compute all pairs (x_P, y_P^2) such that $P = (x_P, y_P)$ is an order-3 point on $\mathcal{F}$. We describe such a procedure below.

1. We first reduce to the depressed quartic polynomial

$$\Delta(z) = z^4 + \left(-\frac{2}{3}A^2 + 2\right) z^2 + \left(\frac{8}{27}A^3 - \frac{4}{3}A\right) z - \frac{1}{27}A^4 + \frac{2}{9}A^2 - \frac{1}{3} \tag{5}$$

   by applying the change of variables $x = z - \frac{1}{3}A$.

2. Next, we calculate the root

$$y = -\frac{1}{9}A^2 + \frac{1}{3} + \frac{1}{3}\sqrt[3]{-2A^2 + 8}$$

   of the cubic polynomial

$$\Gamma(y) = \left(2y - \left(-\frac{2}{3}A^2 + 2\right)\right)\left(y^2 - \left(-\frac{1}{27}A^4 + \frac{2}{9}A^2 - \frac{1}{3}\right)\right) - \frac{1}{4}\left(\frac{8}{27}A^3 - \frac{4}{3}A\right)^2,$$

   which implicitly determines the roots of Equation (5) as follows. Let

$$s_0 = \sqrt{2y - 6\left(-\frac{1}{9}A^2 + \frac{1}{3}\right)},$$

$$s_1 = \sqrt{-2y - 6\left(-\frac{1}{9}A^2 + \frac{1}{3}\right) + \left(\frac{16}{27}A^3 - \frac{8}{3}A\right)\frac{1}{s_0}}, \text{ and}$$

$$s_2 = \sqrt{-2y - 6\left(-\frac{1}{9}A^2 + \frac{1}{3}\right) - \left(\frac{16}{27}A^3 - \frac{8}{3}A\right)\frac{1}{s_0}}.$$

Then the roots of Equation (5) are

$$z_1 = \frac{1}{2}(-s_0 + s_1), \quad z_2 = \frac{1}{2}(-s_0 - s_1), \quad z_3 = \frac{1}{2}(s_0 + s_2), \quad \text{and} \quad z_4 = \frac{1}{2}(s_0 - s_2).$$

3. Finally, the roots of Equation (4) correspond with $x_j = z_j - \frac{1}{3}A$ for each $j \coloneqq 1, 2, 3, 4$.

As a consequence, we can compute all the roots of Equation (4) at the cost of $\mathsf{E}_3 + 3\mathsf{E}_2 + \mathsf{I} + 3\mathsf{M} + 8\mathsf{M}_c + \mathsf{S} + 30\mathsf{A}$ operations. [6] Since the most demanding computations are the cube-root, square-roots and field inversion, this leads to an approximate cost of

$$3\mathsf{E}_2 + \mathsf{E}_3 + \mathsf{I} = (7\mathsf{sqr}_p + 7.5\mathsf{mul}_p)\log_2(p) \approx 14.5\log_2(p)\ \mathsf{mul}_p$$

operations.

Compared to the naive approach of sampling a random point P and calculating $[\frac{1}{3}(p+1)]P$, the above-described deterministic method improves the state of the art. For instance, one randomly samples an element $x_P \in \mathbb{F}_{p^2}$ and checks if $z_P = x_P^3 + ax_P^2 + x_P$ is a quadratic residue in $\mathbb{F}_{p^2}$. Such a check reduces to verifying if $w = (z_1^2 + z_2^2)$ is a quadratic residue in $\mathbb{F}_p$ (e.g., if $w^{(p-1)/2} = 1$ holds), where $z = z_1 + iz_2$ with $z_1, z_2 \in \mathbb{F}_p$. That gives a cost of $\log_2(p)(\mathsf{sqr}_p + \frac{1}{2}\mathsf{mul}_p) = \frac{2}{3}\log_2(p)\mathsf{M}$ multiplications. [7] On the other hand, computing the x-coordinate of $[\frac{1}{3}(p+1)]P$ when given x_P requires $\approx 1.44\log_2(p)$ x-only point additions [CS18]. [8] Since an x-only point addition in Montgomery curve models takes $4\mathsf{M} + 2\mathsf{S} + 6\mathsf{A}$ operations [CS18], this provides a total of $\approx \log_2(p)(5.76\mathsf{M} + 2.88\mathsf{S})$ operations for computing the x-coordinate of $[\frac{1}{3}(p+1)]P$.

However, the probability that a random point P satisfies $[\frac{1}{3}(p+1)]P = \mathbf{0}_{\mathcal{F}}$ is $\frac{1}{5}$, [9] and then this approach has probability of $\frac{4}{5} = 0.8$ to succeed. So, one would need at least two random samplings to ensure a probability of $1 - (\frac{1}{5})^2 = 0.96$ to succeed in finding a root of $\Psi_{\mathcal{F},3}$, at the cost of

$$2\log_2(p)(5.76\mathsf{M} + 2.88\mathsf{S} + \frac{2}{3}\mathsf{M}) = 50.08\log_2(p)\mathsf{mul}_p$$

operations.

Remark 3. The naive approach only works if 3 divides $p+1$, but a minor tweak allows it to extend to the case when 3 divides $p-1$. Conversely, our method works for both cases.

3.3 Computing non-backtracking 3-isogenies

Let us consider the 3-isogeny construction from [CDV20], which is described in Section 2. Let $\mathcal{G}\colon y^2 + a_1xy + a_3y = x^3$ be a supersingular elliptic curve defined over $\mathbb{F}_{p^2}$ and set $\alpha_j = \zeta_3^j(-a_3)^{N_3}$. Notice that α_j determines the three different cube-roots of $-a_3$ (i.e., $\alpha_j^3 = -a_3$). Now, let

$$\mathcal{G}_j\colon y^2 + (-6\alpha_j + a_1)xy + (3a_1\alpha_j^2 - a_1^2\alpha_j + 9a_3)y = x^3 \quad (6)$$

be the 3-isogenous curve to $\mathcal{G}$ for each $j \coloneqq 0, 1, 2$. Certainly, the curves $\mathcal{G}_0$, $\mathcal{G}_1$, and $\mathcal{G}_2$ determine different supersingular elliptic curves. Still, to use them for computing a

[6] Here M_c denotes a multiplication by a constant (e.g., $\frac{1}{2}$, $\frac{1}{3}$, $\frac{1}{9}$, $\frac{1}{27}$).

[7] Such a cost is under the observation that $N = (p-1)/2$ has Hamming weight approximately equal to $\frac{1}{2}\log_2(p)$ for concrete cryptographic parameters.

[8] [CS18] gives the number of steps of the PRAC algorithm [CS18, Alg. 10], which is $\approx 0.72\log_2(p)$. Each step performs a point doubling and point addition, and an x-only point doubling cost $4\mathsf{M} + 2\mathsf{S} + 4\mathsf{A}$ operations.

[9] There are exactly five possibilities for the x-coordinate of a 3-torsion point on $\mathcal{F}$ over $\mathbb{F}_{p^2}$ (the roots of $\Phi_{\mathcal{F},3}(x)$ and the one corresponding to $\mathbf{0}_{\mathcal{F}}$).

non-backtracking 3-isogeny $\vartheta\colon \mathcal{G}_i \to \mathcal{E}_i'$, we must ensure first ϑ determines non-isomorphic codomain curves.

We follow a similar reasoning from [CDV20], but this time, to show why those curves determine non-isomorphic curves. Let us focus on the curve $\mathcal{G}_0$ determined by α_0. By construction, we know that $\mathcal{G}_0$ is isomorphic to the codomain of the 3-isogeny $\vartheta\colon \mathcal{G} \to \mathcal{E}$ with kernel $\ker\vartheta = \langle (0,0)\rangle$. In particular,

$$\mathcal{E}\colon y^2 + a_1xy + a_3y = x^3 - 5a_1a_3x - a_1^3a_3 - 7a_3^2,$$

and its 3-division polynomial is

$$\Psi_{\mathcal{E},3}(x) = 3(x + \frac{1}{3}a_1^2)(x + a_1\alpha_0 - 3\alpha_0^2)(x^2 + (-a_1\alpha_0 + 3\alpha_0^2)x + a_1^2\alpha_0^2 - 3a_1a_3 - 9a_3\alpha_0),$$

such that its roots determine the x-coordinates of order-3 points on $\mathcal{E}$. The first linear factor $(x + \frac{1}{3}a_1^2)$ determines the kernel of the dual 3-isogeny $\hat{\varphi}\colon \mathcal{E} \to \mathcal{G}$. The second linear factor $(x + a_1\alpha_0 - 3\alpha_0^2)$ determines the order-3 point $P_0 = (-a_1\alpha_0 + 3\alpha_0^2, 4a_3)$ on $\mathcal{E}$, which defines the curve $\mathcal{G}_0$ under the translation given by $P_0 \mapsto (0,0)$.

Next, we proceed by showing the curves $\mathcal{G}_1$ and $\mathcal{G}_2$ come from the remaining roots of the last quadratic factor in $\Psi_{\mathcal{E},3}(x)$. Let $x_1 = -a_1\alpha_1 + 3\alpha_1^2$ and $x_2 = -a_1\alpha_2 + 3\alpha_2^2$. Notice that $\alpha_1 = \frac{-1+i\sqrt{3}}{2}\alpha_0$ and $\alpha_2 = \frac{-1-i\sqrt{3}}{2}\alpha_0$. Now, evaluating $f(x) = x^2 + (-a_1\alpha_0 + 3\alpha_0^2)x + a_1^2\alpha_0^2 - 3a_1a_3 - 9a_3\alpha_0$ at x_1 and x_2 gives

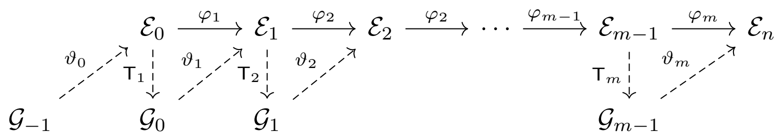
$$f(x_1) = x_1^2 + (-a_1\alpha_0 + 3\alpha_0^2)x_1 - +a_1^2\alpha_0^2 - 3a_1a_3 - 9a_3\alpha_0 = -3(a_3 + \alpha_0^3)(a_1 + 3\alpha_0), \text{ and}$$
$$f(x_2) = x_2^2 + (-a_1\alpha_0 + 3\alpha_0^2)x_2 - +a_1^2\alpha_0^2 - 3a_1a_3 - 9a_3\alpha_0 = -3(a_3 + \alpha_0^3)(a_1 + 3\alpha_0).$$

In other words, $f(x_1) = 0$ and $f(x_2) = 0$ since $\alpha_0^3 = -a_3$. Notice that $P_0 = (-a_1\alpha_0 + 3\alpha_0^2, 4a_3)$, $P_1 = (-a_1\alpha_1 + 3\alpha_1^2, 4a_3)$, and $P_2 = (-a_1\alpha_2 + 3\alpha_2^2, 4a_3)$ define three different order-3 points on $\mathcal{E}$. In addition, translating the order-3 point P_1 (resp. P_2) to $(0,0)$ yields the corresponding curve $\mathcal{G}_1$ (resp. $\mathcal{G}_2$). This implies that each 3-isogeny $\vartheta\colon \mathcal{G}_i \to \mathcal{E}_i'$ with $\ker\vartheta = \langle(0,0)\rangle$ determines a different non-backtracking 3-isogeny.

3.4 Performing length-m 3-isogeny walks with non-backtracking

Let $P = (x_P, y_P)$ be an order-3 point on a Montgomery curve $\mathcal{F}_{-1}\colon y^2 = x^3 + Ax^2 + x$, and let $\mathcal{G}_{-1}\colon y^2 + a_{1,0}xy + a_{3,0}y = x^3$ with $a_{1,0} = (3x_P^2 + 2Ax_P + 1)$ and $a_{3,0} = y_P^4$. Additionally, let $\mathcal{E}_0\colon y^2 + a_{1,0}xy + a_{3,0}y = x^3 - 5a_{1,0}a_{3,0}x - a_{1,0}^3a_{3,0} - 7a_{3,0}^2$ be the codomain of the 3-isogeny $\vartheta_0\colon \mathcal{G}_{-1} \to \mathcal{E}_0$ with $\ker\vartheta_0 = \langle(0,0)\rangle$.

As previously described above, the 3-isogeny formulas from [CDV20] correspond by translating the order-3 point $(-a_{1,0}\alpha_j + 3\alpha_j^2, 4a_{3,0})$ on the curve $\mathcal{E}_0$ into the point $(0,0)$ on $\mathcal{G}_0\colon y^2 + (-6\alpha_j + a_{1,0})xy + (3a_{1,0}\alpha_j^2 - {a_{1,0}}^2\alpha_j + 9a_{3,0})y = x^3$ and then calculating the 3-isogeny $\vartheta\colon \mathcal{G}_0 \to \mathcal{E}_1$ with kernel generated by $(0,0)$, where $\alpha_j = \zeta_3^j\sqrt[3]{-a_{3,0}}$. However, we suggest using the curve model $\mathcal{E}_i\colon y^2 + a_{1,i}xy + a_{3,i}y = x^3 - 5a_{1,i}a_{3,i}x - a_{1,i}^3a_{3,i} - 7a_{3,i}^2$ instead of $\mathcal{G}_i\colon y^2 + a_{1,i}xy + a_{3,i}y = x^3$ to represent the elliptic curves nodes from any 3-isogeny path. In that way, a string $\mathsf{path} \in \{0,1,2\}^m$ determines the length-m 3-isogeny chain $\mathcal{E}_0 \xrightarrow{\varphi_1} \mathcal{E}_1 \xrightarrow{\varphi_2} \cdots \xrightarrow{\varphi_m} \mathcal{E}_m$ described in Equation (7).



(7)

The i^{th} trit of path defines the non-backtracking i^{th} 3-isogeny $\varphi_i = \vartheta_i \circ \mathsf{T}_i$, where $\vartheta_i\colon \mathcal{G}_{i-1} \to \mathcal{E}_i$ is the 3-isogeny with $\ker\vartheta_i = \langle(0,0)\rangle$, and T_i is the translation map given by $(-a_{1,i}\alpha_i + 3\alpha_i^2, 4a_{3,i}) \mapsto (0,0)$ with $\alpha_i = \zeta_3^{\mathsf{path}_i}\sqrt[3]{-a_{3,i}}$.

If one uses the curve model $\mathcal{G}_i\colon y^2 + a_{1,i}xy + a_{3,i}y = x^3$ from [CDV20] to represent the elliptic curves nodes from any 3-isogeny path, then the i^{th} trit of path does not determine the i^{th} 3-isogeny φ_i. In particular, the path describes the length-m 3-isogeny walk $\mathcal{G}_0 \xrightarrow{\varrho_1} \mathcal{G}_1 \xrightarrow{\varrho_2} \cdots \xrightarrow{\varrho_{m-1}} \mathcal{G}_{m-1}$ with $\varrho_i = \mathsf{T}_{i+1} \circ \vartheta_i$. In [NO24], the authors randomly sample an order-3 point $P = (x_P, y_P)$ and perform an length-m 3-isogeny walk of the form $\mathcal{G}_{-1} \to \mathcal{G}_0 \to \cdots \to \mathcal{G}_{m-1}$. However, our analysis refines why such length-m 3-isogeny walks do not have backtracking. In particular, similar to previously mentioned, [NO24] could increase the number of isogenies since the m^{th} trit of the path determines the non-calculate 3-isogeny $\vartheta_m\colon \mathcal{G}_{m-1} \to \mathcal{E}_m$ with $\ker\vartheta_m = \langle(0,0)\rangle$. This gives a space of 3^{m+1} instead of 3^m, which could slightly improve the parameter choice and performance in [NO24].

Remark 4. Notice that the elliptic curves $\mathcal{E}_i\colon y^2+a_{1,i}xy+a_{3,i}y = x^3-5a_{1,i}a_{3,i}x-a_{1,i}^3a_{3,i}-7a_{3,i}^2$ are described only by the two coefficients $a_{1,i}$ and $a_{3,i}$. Therefore, our curve model suggestion requires exactly the same number of elements in $\mathbb{F}_{p^2}$ (the two elements $a_{1,i}$ and $a_{3,i}$) than using $\mathcal{G}_i\colon y^2 + a_{1,i}xy + a_{3,i}y = x^3$ to describe the elliptic curve nodes from any 3-isogeny path.

3.5 CGL Hash function built on top of 3-isogenies

As implicitly highlighted before, and opposite to $\mathsf{CGLHash}_2$, we suggest that each isogeny step in $\mathsf{CGLHash}_3$ calculates 3-isogenies between curves of the form $\mathcal{E}\colon y^2 + a_1xy + a_3y = x^3-5a_1a_3x-a_1^3a_3-7a_3^2$. This, in some sense, is slightly different from $\mathsf{CGLHash}_2$ where the isogeny calculations are via the 2^{nd} modular polynomial $\Phi_2(x,y)$ and, therefore, directly on the j-invariants of the curves.

In a nutshell, given an arbitrary supersingular elliptic curve $\mathcal{F}\colon y^2 = x^3 + Ax^2 + x$ and the x-coordinate x_P of an order-3 point P on $\mathcal{F}$, we describe the $\mathsf{CGLHash}_3$ as follows. Let $a_1 = (3x_P^2 + 2Ax_P + 1)$ and $a_3 = 2y_P^4 = 2(x_P^3 + Ax_P^2 + x_P)^2$ such that describe the supersingular elliptic curve $\mathcal{E}\colon y^2 + a_1xy + a_3y = x^3 - 5a_1a_3x - a_1^3a_3 - 7a_3^2$.

An efficient $\mathsf{CGLHash}_3$ function description:

Step 1: Take as input a trit-string path $\in \{0,1,2\}^*$ of length m.

Step 2: Compute the length-m 3-isogeny walk $\mathcal{E}_0 \coloneqq \mathcal{E} \xrightarrow{\varphi_1} \mathcal{E}_1 \xrightarrow{\varphi_2} \cdots \xrightarrow{\varphi_n} \mathcal{E}' \coloneqq \mathcal{E}_n$ determined by path as below.

- *Step 2.1:* Set $a_1' \leftarrow a_1$ and $a_3' \leftarrow a_3$.
- *Step 2.2:* For each $i \coloneqq 1,\ldots,n$.
  - *Step 2.2.1:* Calculate $\alpha = \zeta_3^{\mathsf{path}_i}\sqrt[3]{-a_3}$.
  - *Step 2.2.2:* Update $a_1' \leftarrow -6\alpha + a_1'$.
  - *Step 2.2.3:* Update $a_3' \leftarrow 3a_1'\alpha^2 - {a_1'}^2\alpha + 9a_3'$.

Step 3: Outputs the j-invariant of $\mathcal{E}'\colon y^2+a_1'xy+a_3'y = x^3-5a_{1'}a_{3'}x-a_1'^3a_3'-7a_3'^2$, which is equal to

$$j(\mathcal{E}') = \frac{a_1'^3\left(a_1'^3 + 216a_3'\right)^3}{a_3'\left(a_1'^3 - 27a_3'\right)^3}. \tag{8}$$

3.6 On the implications to QFESTA

In [NO24], the authors propose prime numbers of the form $p = 2^{3a} \cdot 3 \cdot f - 1$ with $\gcd(3,f) = 1$, restricting to the scenario when 3 divides $(p+1)$. However, our results are not limited to that constraint and hold for any prime in general, but for efficiency, we assume p such that $(p^2-1) = 2^{3a} \cdot 3 \cdot f$ and $\gcd(3, f) = 1$.

For instance, [NO24] takes the conservative choice of $a = \lambda + 2$, and suggests the following parameter sets for $\lambda \in \{128, 192, 256\}$.

- QFESTA-128: $p398 = 2^{3\cdot 130} \cdot 3 \cdot 55 - 1$ of 398 bits.
- QFESTA-192: $p592 = 2^{3\cdot 194} \cdot 3 \cdot 307 - 1$ of 592 bits.
- QFESTA-256: $p783 = 2^{3\cdot 258} \cdot 3 \cdot 137 - 1$ of 783 bits.

Those parameter sets do not take into account possible optimizations for the prime bitlengths. In practice, one would opt for primes with $\log_2(p) \lessapprox 3\lambda$ since $64 \mid \lambda$, which allows a more efficient arithmetic implementation in terms of bit operations. In addition, the security analysis of [NO24] highlights a time complexity for breaking QFESTA of at least $\tilde{O}(2^\lambda)$. Therefore, one can slightly relax the size of a, and still provide the desired bit security.

For example, as a less conservative choice, we list the following parameter sets for QFESTA. [10]

- QFESTA-128: $p381 = 2^{3\cdot 124} \cdot 437 - 1$ of 381 bits.
- QFESTA-192: $p575 = 2^{3\cdot 189} \cdot 139 - 1$ of 575 bits.
- QFESTA-256: $p765 = 2^{3\cdot 252} \cdot 257 - 1$ of 765 bits.

These parameter sets have one 64-bit word less than the proposed in [NO24], which could benefit from a faster field arithmetic implementation. However, further analysis is required for the usage of such a parameter sets, and, therefore, we limit our analysis to illustrate the practical advantage of these less conservative parameter sets in Section 4.

3.7 On the implications to CSIDH

As mentioned in the introduction, the original case of use for radical 3-isogeny formulas centers on performing chains of 3-isogenies over the base field $\mathbb{F}_p$, hoping to speed up the CSIDH protocol.

In CSIDH, one needs to compute m ℓ-isogenies with ℓ being a small odd prime; in particular, some variants of CSIDH (e.g., CTIDH [BBC+21] [11]) require sampling random points, which, as mentioned before, are expensive compared to the cost of calculating ℓ-isogenies. While other variants of CSIDH (e.g., dCSIDH [CCSC+24]) exclude the usage of 3-isogenies to avoid the expensive cost of the sampling point. With the expensive cost of sampling points in mind, the formulas from [CDV20] remove such a sampling per ℓ-isogeny (the formulas only require a single sampling point per 3-isogeny chain, that is, for a bunch of 3-isogenies!).

General considerations for CSIDH: In general, we explicitly require knowing which order-3 point belongs to $\mathcal{E}[\pi-1]$ and which one to $\mathcal{E}[\pi+1]$; more precisely, if $x_P \in \mathbb{F}_p$ determines the x-coordinate of a point P on $\mathcal{E}$, we need to determine if $w_P = (x_P^3 + Ax_P^2 + x_P)$ is a quadratic residue (resp. a non-quadratic residue) corresponding to a point on $\mathcal{E}[\pi-1]$ (resp. on $\mathcal{E}[\pi+1]$). Based on these considerations and employing the technique from Section 3.2, we end up with a cost of $6\mathsf{exp}_p + 3\mathsf{mul}_p + \mathsf{sqr}_p$ operations for calculating two x-only order-3

[10] We construct these proposed prime numbers by using the script `parameter_generate.py` from https://github.com/hiroshi-onuki/QFESTA-SageMath, and removing the factor of three from Line 119 of `parameter_generate.py`.

[11] CTIDH is the current most efficient variant of CSIDH, and it still uses 3-isogenies.

points (one in $\mathcal{E}[\pi-1]$ and another in $\mathcal{E}[\pi+1]$) from a given Montgomery curve coefficient $A \in \mathbb{F}_p$.

Given that any variant of the CSIDH protocol works explicitly over the Montgomery curve model, taking the Montgomery curve coefficient $A \in \mathbb{F}_p$ as inputs and outputs, we opt for using the formulas from [OM22] given in the Montgomery curve model. However, in practice, one works with projective Montgomery curve coefficients $(A+2C : 4C) \in \mathbb{P}^1(\mathbb{F}_p)$ and kernel point generators $(X : Y) \in \mathbb{P}^1(\mathbb{F}_p)$ to avoid calculating field inversions, which is not the case for the formulas in [OM22].

Therefore, let us describe a projective variant approach of our procedure in Section 3.2 and subsequently introduce a projective version of the formulas from [OM22]. As the main ingredient for achieving such projective variants, we present the following lemma. [12]

Lemma 1. *Let $n \in \mathbb{N}$. Let $\alpha = X/Z \in \mathbb{F}_{p^2}$ with $X, Z \in \mathbb{F}_{p^2}$ such that $Z \neq 0$. Then $\beta = \sqrt[n]{XZ^{n-1}}/Z$ satisfies $\beta^n = \alpha$.*

Proof. Observe that $\alpha = X/Z = (XZ^{n-1})/(Z^n)$. Hence, $\beta = \sqrt[n]{XZ^{n-1}}/Z$ satisfies $\beta^n = X/Z$. In particular, it follows that α has an n^{th} root if and only if XZ^{n-1} so does. ☐

Computing x-coordinates of order-3 points without field inversions: Let $\mathcal{E} : y^2 = x^3 + Ax^2 + x$ with $A \in \mathbb{F}_p$. According to Section 3.2, the four order-3 points on $\mathcal{E}$ have the x-coordinates $x_j = z_j - \frac{1}{3}A$ where

$$z_1 = \frac{1}{2}(-s_0 + s_1), \quad z_2 = \frac{1}{2}(-s_0 - s_1), \quad z_3 = \frac{1}{2}(s_0 + s_2), \quad \text{and} \quad z_4 = \frac{1}{2}(s_0 - s_2)$$

with s_0, s_1, s_2 and y defined as in Section 3.2. Let us write $A = A'/C$ for some $A', C \in \mathbb{F}_p$ and $C \neq 0$. Then, applying Lemma 1 with $n = 3$ in the cube root in y gives

$$y = \frac{1}{C^2}\underbrace{\left(-\frac{1}{9}(A')^2 + \frac{1}{3}C^2 + \frac{1}{3}\sqrt[3]{(-2(A')^2 + 8C^2)C^4}\right)}_{y'}.$$

Next, applying Lemma 1 with $n = 2$ in the square root in s_0 gives

$$s_0 = \frac{1}{C}\underbrace{\sqrt{2y' - 6\left(-\frac{1}{9}(A')^2 + \frac{1}{3}C^2\right)}}_{s_0'}.$$

Similarly, applying Lemma 1 with $n = 2$ but this time in the square roots in s_1 and s_2 gives

$$s_1 = \frac{1}{Cs_0'}\overbrace{\sqrt{\left(-2y' - 6\left(-\frac{1}{9}(A')^2 + \frac{1}{3}C^2\right)\right)(s_0')^2 + \left(\frac{16}{27}(A')^3 - \frac{8}{3}A'C^2\right)s_0'}}^{s_1'}, \text{ and}$$

$$s_2 = \frac{1}{Cs_0'}\underbrace{\sqrt{\left(-2y' - 6\left(-\frac{1}{9}(A')^2 + \frac{1}{3}C^2\right)\right)(s_0')^2 - \left(\frac{16}{27}(A')^3 - \frac{8}{3}A'C^2\right)s_0'}}_{s_2'}.$$

Hence, it follows that $x_j = X_j/Z_j$ where $Z_j = Cs_0'$, $X_j = z_j' - \frac{1}{3}A's_0'$, and

$$z_1' = \frac{1}{2}(-(s_0')^2 + s_1), \quad z_2' = \frac{1}{2}(-(s_0')^2 - s_1), \quad z_3' = \frac{1}{2}((s_0')^2 + s_2), \quad z_4' = \frac{1}{2}((s_0')^2 - s_2).$$

[12] Lemma 1 can be viewed as a straightforward generalization of [CR22, Lemma 3.1].

Notice that there are two possible options: 1) $s'_1 \in \mathbb{F}_p$ and $s'_2 \in i\mathbb{F}_p$, or 2) $s'_1 \in i\mathbb{F}_p$ and $s'_2 \in \mathbb{F}_p$. Therefore, as previously mentioned, the choice between s'_1 and s'_2 can be done by checking if s''_1 is a quadratic residue (or not) where $s'_1 = \sqrt{s''_1}$.

Thereafter, the cost of computing both order-3 points in $\mathcal{E}[\pi - 1]$ and $\mathcal{E}[\pi + 1]$ reduces to $(5\mathsf{exp}_p + 15\mathsf{mul}_p + 8\mathsf{mulc}_p + 6\mathsf{sqr}_p) \approx 7.5\log_2(p)$ mul_p operations [13] when outputting projective x-only points from a projective curve coefficient as input. This is a considerable improvement compared with the state-of-the-art where sampling a 3-torsion basis is required; for example, in average $4 \cdot (6.26\mathsf{mul}_p + 3.88\mathsf{sqr}_p)\log_2(p) \approx 40.56\log_2(p)$ mul_p operations are needed according to Section 3.2.

Projective radical 3-isogeny formulas: Let (t, y) be an order-3 point on $\mathcal{E}$ such that $(t, y) \in \mathcal{E}[\pi - \epsilon]$ with $\epsilon \in \{1, -1\}$. Recall that $t' = 3t\alpha^2 + (3t^2 - 1)\alpha + 3t^3 - 2t$ determines the x-coordinate of an order-3 point $(t', y') \in \mathcal{E}'[\pi - \epsilon]$ where $\mathcal{E}' \colon y^2 = x^3 + A'x^2 + x$ is the codomain of the isogeny with kernel generated by (t, y) and $\alpha = \sqrt[3]{t(t^2 - 1)}$ [OM22]. Now, let us assume $t = R/S$ for some $R, S \in \mathbb{F}_p$ such that $S \neq 0$. Then,

$$\alpha = \frac{1}{S}\overbrace{\sqrt[3]{R(R^2 - S^2)}}^{\alpha'}, \text{ and}$$
$$t' = \frac{1}{S^3}\underbrace{\left(3R(\alpha')^2 + (3R^2 - S^2)\alpha' + 3R^3 - 2RS^2\right)}_{R'}.$$

Hence, we can perform m_3 projective radical 3-isogenies over an arbitrary Montgomery curve at the cost of

$$\underbrace{5\ \mathsf{exp}_p + 15\mathsf{mul}_p + 6\mathsf{sqr}_p}_{\text{3-torsion basis}} + \underbrace{m_3(\mathsf{exp}_p + 6\mathsf{mul}_p + 3\mathsf{sqr}_p)}_{\text{Radical 3-isogenies}}$$

operations, which is approximately equal to

$$(1.5 \cdot m_3 + 7.5)\log_2(p) + (21 + 9 \cdot m_3)$$

multiplications over $\mathbb{F}_p$. In addition, we can retrieve the Montgomery curve coefficient A' at the cost of $\mathsf{inv}_p + 2\mathsf{mul}_p + 4\mathsf{sqr}_p$ operations as follows.

$$A' = \frac{-3(t')^4 - 6(t')^2 + 1}{4(t')^3} = \frac{-3(R')^4 - 6(R')^2(S')^2 + (S')^4}{4(R')^3 S'},$$

where $S' = S^3$.

Specific considerations for CTIDH: Plugging our results into the CTIDH protocol requires more than replacing the 3-isogeny formulas; it requires a thoughtful analysis for determining optimal parameters (i.e., private keyspace). In particular, the CTIDH authors emphasize that searching for optimal parameters is challenging [BBC+21]. However, considering that the keyspace of CTIDH is approximately of size 2^n, adding $m_3 = 2^{m-1}$ radical 3-isogenies increases such keyspace to a size of $(2^m + 1) \times 2^n$ secret keys. Therefore, we can decrease the value of n by m without any loss of security, which gives the following keyspace size.

$$\underbrace{(2^m + 1)}_{\text{3-isogeny part}} \times \overbrace{2^{n-m}}^{\text{CTIDH part}} \approx 2^n.$$

We illustrate the practical advantage of using the this keyspace modification in Section 4.

[13] Here mulc_p denotes a multiplication by a constant (e.g., $\frac{1}{2}$, $\frac{1}{3}$, $\frac{1}{9}$, $\frac{1}{27}$)

4 Comparisons and Experiments

Our results from Section 3 highlight the main advantage of using 3-isogenies: we can perform shorter length-m 3-isogeny walks than length-n 2-isogenies walks, where $m = n/\log_2(3)$. We emphasize that the main advantage of our results centers when there is a need for calculating 3-isogenies as required in [NO24, BBC+21, CHMR25].

We provide a dedicated implementation in C-language to experimentally validate the advantage of using 3-isogenies instead of 2-isogenies. Given the structure of the $\mathsf{CGLHash}_3$ function from Section 3.5, we highlight that implementing it in constant-time [14] becomes straightforward by calculating $\alpha = \zeta_3^{\mathsf{path}_i}\sqrt[3]{-a_3}$ through a linear-pass procedure on the list $[\sqrt[3]{-a_3}, \zeta_3\sqrt[3]{-a_3}, \zeta_3^2\sqrt[3]{-a_3}]$. Similarly for $\mathsf{CGLHash}_2$ (see Equation (3)), we provide a constant-time implementation of it. Additionally, the field arithmetic implementation is based on the optimized implementation from [BBC+21]. We emphasize that we provide two dedicated implementations:

1. An implementation that performs 3-isogeny random walk over $\mathbb{F}_{p^2}$ (as required in [NO24]).
2. An implementation that calculates chains of 3-isogenies over $\mathbb{F}_p$, with the goal of improving [BBC+21, CCSC+24, CHMR25].

Our testbed consists of machine with a 12th Gen. Intel(R) Core(TM) i9-12900H CPU and 32 Gb of RAM, running Ubuntu 24.04.2 LTS (64 bits). To avoid the operating system to use multi-thread or additional performance boosts, we disabled the Turbo Boost, hyperthreading, and perform the calculations in one single CPU.

Benchmarking CPU cycles and Execution Time. We perform two types of benchmarking: CPU cycles and Execution Time. To measure the CPU cycle count between executions of our tests, we use the **rdtsc** assembly instruction for Intel architectures, returning an integer value of cycles. The execution time was measured in nanoseconds (ns). As both the CPU cycle count and execution time show the same behavior, in the manuscript we only include the CPU cycle count graphs.

4.1 Experiments over $\mathbb{F}_{p^2}$

Setup for the case $p^2 \equiv 9 \bmod 16$. Our experiments focus on the 511-bit prime number $p511$ from [CLM+18]. Additionally, we incorporate experiments for two prime numbers $p255$ and p383 constructed following the CSIDH form of $p511$. More precisely,

$$p255 = 4 \cdot \frac{281}{17} \cdot \prod_{i=1}^{41} \ell_i - 1 \quad \text{and} \quad p383 = 4 \cdot \frac{367}{5} \cdot \prod_{i=1}^{57} \ell_i - 1,$$

where $\ell_1, \ldots, \ell_{57}$ correspond with the first 57 small odd prime numbers. We opt for including these primes given that the radical isogeny formulas from [CDV20] were originally proposed for CSIDH-like primes.

Setup for the case $p^2 \not\equiv 9 \bmod 16$. We additionally include experiments for the scenario when $p^2 \not\equiv 9 \bmod 16$. In particular, the field characteristic is of the form $p = 2^a \cdot f - 1$ for some positive integers $a > 2$ and c. Our experiments centers on the primes $p398$, $p592$, and $p783$ from [NO24], and the prime $p254$ from [DMPR24]. In addition, we include experiments

[14] By constant-time we mean the running time of an algorithm is independent from its input. In particular, our implementation protects against timing attacks by avoiding secret-dependent branch conditions.

considering our suggested primes $p381$, $p575$, and $p765$ from Section 3.6. In all our experiments, we calculate length-n 2-isogeny chains with $n = 2\lambda$, where $\lambda \in \{128, 192, 256\}$; thus, we compute length-m 3-isogeny chains with $m = \left\lfloor \frac{2}{\log_2(3)} \lambda \right\rceil$. Table 3 lists the used values of n and m in our experiments.

Table 3: Lengths of the 2-isogeny (value of n) and 3-isogeny (value of m) walks.

p:	$p254$	$p255$	$p381$	$p383$	$p398$	$p511$	$p575$	$p592$	$p765$	$p783$
n:	256	256	256	384	256	512	384	384	512	512
m:	162	162	162	243	162	324	243	243	324	324

Numerical Results over $\mathbb{F}_{p^2}$. We performed benchmarks for the traditional 2-isogenies walks and our 3-isogenies walks. These benchmarks included measurements made in CPU cycles. Figure 3 shows a summary of the results obtained after running ten thousand $(10,000)$ independent walks $(10,000$ independent 2-isogenies walk and $10,000$ independent 3-isogenies walk per prime) in a single-core CPU (no hyperthreading) and no Turbo Boost scenario. Our experiments illustrates that performing 3-isogenies walks of length $m = n/\log_2 3$ is competitive compared with 2-isogeny walks of length n. We reiterate that the main advantage of our results centers when there is a need to computing 3-isogenies walks as required in [NO24, BBC+21, CHMR25], and our comparisons against 2-isogeny walks are merely illustrative.

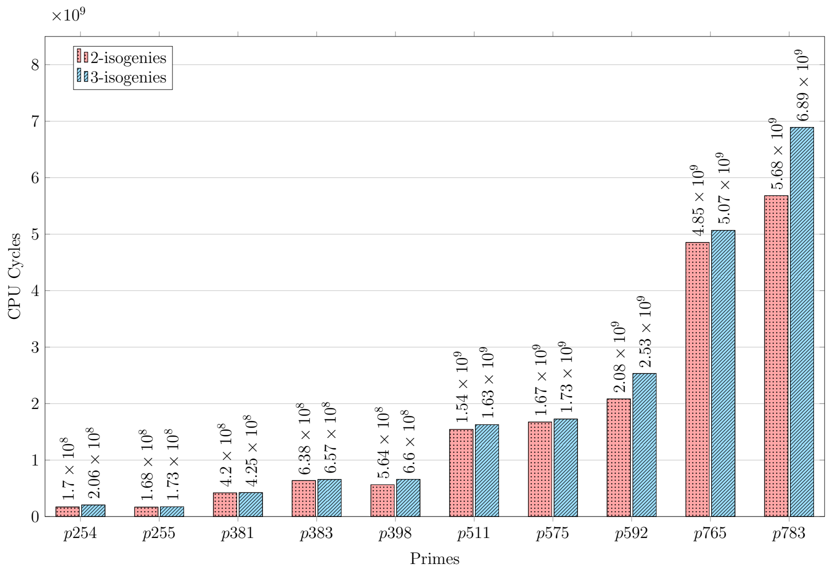


Figure 3: Benchmarks for the 2-isogenies vs. 3-isogenies walks, measured in CPU cycles.

Additionally, we prove by numbers that our suggested primes ($p381$, $p575$ and $p765$) provide a better performance in the 3-isogenies walk scenario compared to the primes ($p398$, $p592$ and $p783$) proposed by QFESTA [NO24]. Both primes $p381$ and $p398$ offer 128-bits security, while $p575$ and $p592$ offer 192-bits security, and $p765$ and $p783$ offer 256-bits security. Similar to the experiments results shown in Figure 3, we performed the CPU cycles and execution time benchmarking of these three pairs of primes. Figure 4 shows that our proposed primes have an improvement in performance of 35.60% for 128-bits, 31.62% for 192-bits, and 26.41% for 256-bits, respectively, with respect to CPU-cycles.

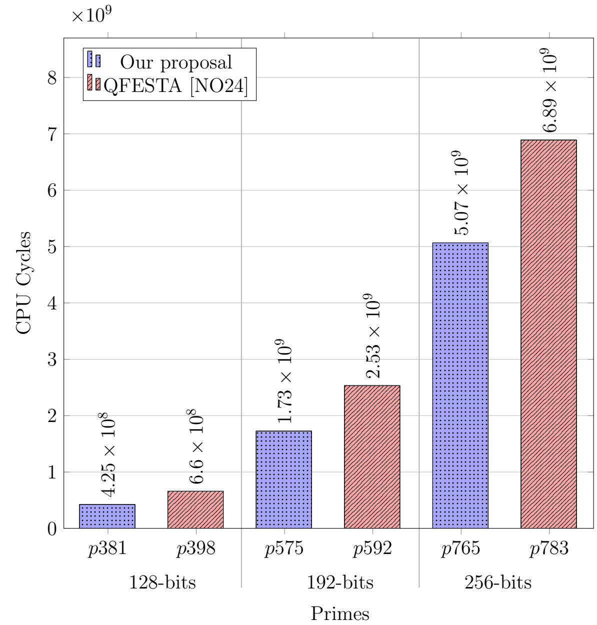


Figure 4: Benchmarks for the 3-isogenies walks for our proposed primes ($p381$, $p575$ and $p765$) vs. QFESTA [NO24] primes ($p398$, $p592$ and $p783$). Both $p381$ and $p398$ offer 128-bits security, while $p575$ and $p592$ offer 192-bits security, and $p765$ and $p783$ offer 256-bits security. For these six primes, the performance was measured in CPU cycles, having an improvement of 35,60% for 128-bits, 31.62% for 192-bits, and 26.41% for 256-bits, respectively.

4.2 Experiments over $\mathbb{F}_p$

Setup for CTIDH experiments. To illustrate the impact of our formulas in the CTIDH context, we center our experimental analysis on the most efficient CTIDH variant (i.e., dCTIDH [CHMR25]). In particular, we take the parameter set from [CHMR25] determined by the following 2048-bits prime numbers:

$$p194 = 2^{387} \cdot 287 \cdot \prod_{j=1}^{194} \ell_i - 1, \quad p205 = 2^{275} \cdot 221 \cdot \prod_{j=1}^{205} \ell_i - 1, \quad \text{and} \quad p226 = \frac{2^{64}}{1361} \cdot \prod_{j=1}^{226} \ell_i - 1,$$

where $\ell_1, \ldots, \ell_{226}$ are the first 226 small odd prime numbers.

Remark 5. It could be for sure a bit worrying to see the expensive cost of performing radical 3-isogeny chains; likewise, we must take into consideration that in higher security parameter sets for dCTIDH: the last isogeny have a degree greater than 1400, and adding a new ℓ_{n+1}-isogeny with $\ell_{n+1} > 1400$ implies adding the calculation of the codomain Montgomery curve coefficient concerning ℓ_{n+1}, and (at least) pushing two points through the ℓ_n-isogeny. Thus, adding such a $(n+1)^{\text{th}}$ isogeny includes around

$$22\left(\sqrt{\ell_{n+1}}\right)^{\log_2 3} + 30\left(\sqrt{\ell_n}\right)^{\log_2 3} > 52 \cdot \left(\sqrt{1400}\right)^{\log_2 3} \approx 16190 \approx 8 \cdot 2048$$

multiplications when employing the formulas from [BFLS20]. [15] Consequently. adding m new primes $\ell_{n+1}, \ldots \ell_{n+m}$ implies an increased cost of at least $8 \cdot 2048 \cdot m = 8m \log_2(p)$ multiplications. Therefore, we only take into consideration values of $m_3 = 2^{m-1}$ such that

$$(1.5 \cdot 2^{m-1} + 7.5)\log_2(p) + (21 + 9 \cdot 2^{m-1}) < 8m \log_2(p).$$

[15] According to the cost analysis from [CSOPM24, Section 2.3].

Following the above observation, we consider the next keyspace sizes: $(2^m+1)\times 2^{221-m}$ (this corresponds with including $m_3 = 2^{m-1}$ radical 3-isogenies) for each $m = 1, 2, 3, 4, 5$. More precisely, we take the dCTIDH keyspaces from [CHMR25] as a baseline, and decrease the number of ℓ's by m for each choice of m. For $m_3 = 2^{m-1}$ we exclude the values of $\ell \in \{\ell_1 = 3, \ell_2 = 5, \ldots, \ell_m\}$. That is,

- For $m = 1$ we exclude $\ell = 3$.
- For $m = 2$ we exclude $\ell = 3, 5$.
- For $m = 3$ we exclude $\ell = 3, 5, 7$.
- For $m = 4$ we exclude $\ell = 3, 5, 7, 11$.
- For $m = 5$ we exclude the $\ell = 3, 5, 7, 11, 13$.

Given that the key generation of dCTIDH requires finding a full torsion basis of $\mathcal{E}[\ell_1 \cdots \ell_n]$, then removing the smallest ℓ's implies an improvement on the performance of the key generation of dCTIDH. We experimentally verify such assertion: our experiments reflect considerable savings in the key generation (keygen) of dCTIDH, with a small impact on the shared key derivation (derive) of dCTIDH.

Remark 6. We provide a constant-time implementation of 3-isogeny chains in the sense i) it exactly performs 2^{m-1} 3-isogenies, and ii) it follows the dummy-based approach of CSIDH and dCTIDH.

Numerical Results over $\mathbb{F}_p$. Our experiments in $\mathbb{F}_p$ include the efficient implementation of our formulas and their integration to the dCTIDH protocol. from [CHMR25]. Figure 5 shows both the timings of the keygen and the derive procedures of dCTIDH. The timings corresponding with the dCTIDH keygen procedure concerns the isogeny calculations along with the search of the full-torsion point basis as required in dCTIDH, while the dCTIDH derive procedure only incorporates the isogeny calculations. In all our experiments, we take as full-torsion point a point of order

$$d = \frac{\prod_{i=1}^{n} \ell_n}{\ell_1 \cdots \ell_m} \quad \text{with} \quad n \in \{194, 205, 226\}.$$

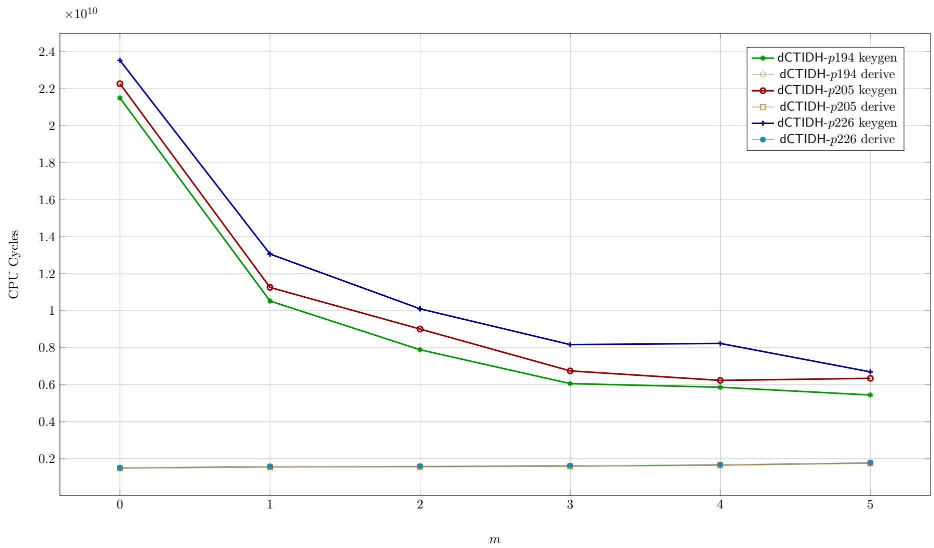
Our experiments illustrate that the key generation of dCTIDH can be as much as close to 4x faster than the original dCTIDH, with a slightly small overhead in the shared key derivation of dCTIDH. The results of each parameter set correspond by computing the mean from 100 random instances. To minimize biases from background tasks running on the benchmark platform, each instance has been repeated ten times and averaged. Tables 4 and 5 show the numerical results of the benchmarks for the keygeneration and derive, respectively, and Figure 5 graphically presents these aforementioned values.

Table 4: CPU cycles (in the order of millions, i.e., 10^6) for the key generation (keygen) reported by the dCTIDH benchmarking tool. For $m = 0$, the original dCTIDH was run, while $m > 0$ implies the use of 2^{m-1} radical 3-isogenies. From these values, we can see that $m = 5$ represents a performance boost of 4.0 times for $p194$, and 3.5 times faster for $p205$ and $p226$.

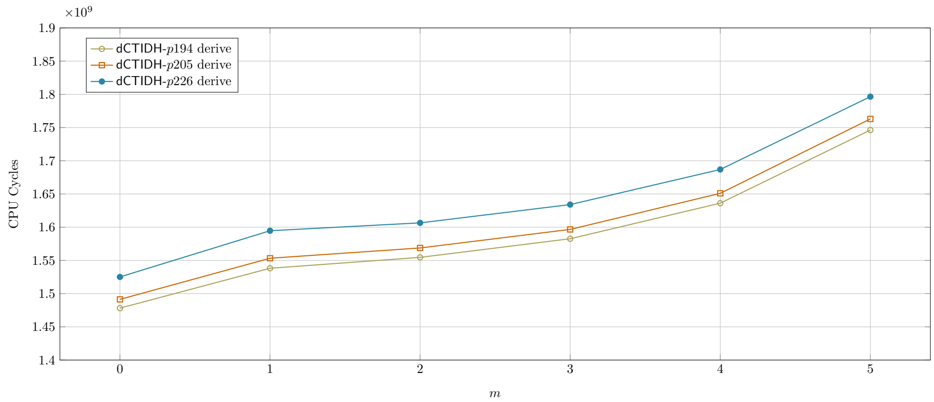
	$m = 0$	$m = 1$	$m = 2$	$m = 3$	$m = 4$	$m = 5$
$p194$	21504.9302	10529.1717	7893.4182	6065.7520	5868.6174	5449.2955
$p205$	22272.8744	11264.1632	9012.5654	6752.0274	6238.9333	6349.9610
$p226$	23534.7420	13067.9704	10105.5958	8172.4838	8236.8301	6699.5763

Table 5: CPU cycles (in the order of millions, i.e., 10^6) for the key derivation (act) reported by the dCTIDH benchmarking tool. For $m = 0$, the original dCTIDH was run, while $m > 0$ implies the use of 2^{m-1} radical 3-isogenies.

	$m = 0$	$m = 1$	$m = 2$	$m = 3$	$m = 4$	$m = 5$
$p194$	1478.1633	1538.1753	1554.5752	1582.6123	1636.1530	1746.2648
$p205$	1491.3575	1553.2523	1568.7469	1596.6670	1650.9843	1762.9833
$p226$	1525.1140	1594.6851	1606.5136	1634.0456	1686.8832	1796.4955



(a)



(b)

Figure 5: Benchmarks for state-of-the-art dCTIDH vs. dCTIDH modified using our proposal. Both the key generation (keygen) and the shared key derivation (derive) were tested. From (a), it is evident that the usage of 3-isogenies creates a huge decrease in the execution time of the key generation, and this impact depends on the number of radical 3-isogenies used. When $m = 5$ radical 3-isogenies are used, the key generation performs 4.0 times faster for $p194$, and 3.5 times faster for both $p205$ and $p226$, respectively. Regarding (b), the use of 3-isogenies introduce a small latency in the key derivation (derive), which is negligible compared to the boost gained in the key generation.

5 Concluding Remarks

This work introduces a novel and efficient method to compute all order-3 points on arbitrary supersingular Montogomery curves. Our results show a considerable improvement in the computation of those order-3 points, where the cost goes from $50\log_2(p)\mathsf{mul}_p$ per point to $14.5\log_2(p)\mathsf{mul}_p$ for computing the four order-3 points for curves over $\mathbb{F}_{p^2}$; and from $40.56\log_2(p)\mathsf{mul}_p$ per point to $7.5\log_2(p)\mathsf{mul}_p$ for computing the two order-3 x-only points for curves over $\mathbb{F}_p$.

This work presents explicit formulas to compute length-m 3-isogeny chains with non-backtracking via the 3-isogeny over $\mathbb{F}_{p^2}$, and novel explicit formulas to compute projective radical 3-isogenies over $\mathbb{F}_p$.

We show that by using radical 3-isgogenies is possible to propose less conservative parameter sets for instantiating [NO24], improving the finite field arithmetic and the isogeny walks computation. Our results show that using this approach we could improve QFESTA up to 35.60%. Our results also offer an excellent opportunity to improve cryptographic schemes where the computation of an order-3 point is needed; we show the applicability of our results in the CTIDH protocol from [BBC+21, CCSC+24], where we obtained a speed-up of up to 4x faster compared to the original implementation.

Finally, we provide the first optimized C-code implementation for computing 3-isogeny walks on arbitrary supersingular Montgomery curves defined over $\mathbb{F}_{p^2}$ and $\mathbb{F}_p$ via cube-root calculations.

References

[BBC+21] Gustavo Banegas, Daniel J. Bernstein, Fabio Campos, Tung Chou, Tanja Lange, Michael Meyer, Benjamin Smith, and Jana Sotáková. CTIDH: faster constant-time CSIDH. *IACR TCHES*, 2021(4):351–387, 2021.

[BFLS20] 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.

[CCSC+24] Fabio Campos, Jorge Chávez-Saab, Jesús-Javier Chi-Domínguez, Michael Meyer, Krijn Reijnders, Francisco Rodríguez-Henríquez, Peter Schwabe, and Thom Wiggers. Optimizations and practicality of high-security CSIDH. *CiC*, 1(1):5, 2024.

[CD20] Wouter Castryck and Thomas Decru. CSIDH on the surface. In Jintai Ding and Jean-Pierre Tillich, editors, *Post-Quantum Cryptography - 11th International Conference, PQCrypto 2020*, pages 111–129. Springer, Cham, 2020.

[CDHV22] Wouter Castryck, Thomas Decru, Marc Houben, and Frederik Vercauteren. Horizontal racewalking using radical isogenies. In Shweta Agrawal and Dongdai Lin, editors, *ASIACRYPT 2022, Part II*, volume 13792 of *LNCS*, pages 67–96. Springer, Cham, December 2022.

[CDV20] Wouter Castryck, Thomas Decru, and Frederik Vercauteren. Radical isogenies. In Shiho Moriai and Huaxiong Wang, editors, *ASIACRYPT 2020, Part II*, volume 12492 of *LNCS*, pages 493–519. Springer, Cham, December 2020.

[CHMR25] Fabio Campos, Andreas Hellenbrand, Michael Meyer, and Krijn Reijnders. *dCTIDH: Fast & Deterministic CTIDH.* Cryptology ePrint Archive, Report 2025/107, 2025.

[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.

[CLL23] Kelong Cong, Yi-Fu Lai, and Shai Levin. Efficient isogeny proofs using generic techniques. In Mehdi Tibouchi and Xiaofeng Wang, editors, *ACNS 23International Conference on Applied Cryptography and Network Security, Part II*, volume 13906 of *LNCS*, pages 248–275. Springer, Cham, June 2023.

[CLM+18] Wouter Castryck, Tanja Lange, Chloe Martindale, Lorenz Panny, and Joost Renes. CSIDH: An efficient post-quantum commutative group action. In Thomas Peyrin and Steven Galbraith, editors, *ASIACRYPT 2018, Part III*, volume 11274 of *LNCS*, pages 395–427. Springer, Cham, December 2018.

[CR22] Jesús-Javier Chi-Domínguez and Krijn Reijnders. Fully projective radical isogenies in constant-time. In Steven D. Galbraith, editor, *CT-RSA 2022*, volume 13161 of *LNCS*, pages 73–95. Springer, Cham, March 2022.

[CS18] Craig Costello and Benjamin Smith. Montgomery curves and their arithmetic - the case of large characteristic fields. *Journal of Cryptographic Engineering*, 8(3):227–240, September 2018.

[CSOPM24] Jorge Chávez-Saab, Odalis Ortega, and Amalia Pizarro-Madariaga. On the parallelization of square-root Vélu's formulas. Cryptology ePrint Archive, Report 2024/851, 2024.

[CSRT22] Jorge Chávez-Saab, Francisco Rodríguez-Henríquez, and Mehdi Tibouchi. Verifiable isogeny walks: Towards an isogeny-based postquantum VDF. In Riham AlTawy and Andreas Hülsing, editors, *SAC 2021*, volume 13203 of *LNCS*, pages 441–460. Springer, Cham, September / October 2022.

[Dec24] Thomas Decru. Radical $\sqrt[N]{\text{élu}}$ isogeny formulae. In Reyzin and Stebila [RS24], pages 107–128.

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

[DMPR24] Pierrick Dartois, Luciano Maino, Giacomo Pope, and Damien Robert. An algorithmic approach to (2, 2)-isogenies in the theta model and applications to isogeny-based cryptography. In Kai-Min Chung and Yu Sasaki, editors, *ASIACRYPT 2024, Part III*, volume 15486 of *LNCS*, pages 304–338. Springer, Singapore, December 2024.

[KCYL06] Fanyu Kong, Zhun Cai, Jia Yu, and Daxing Li. Improved generalized Atkin algorithm for computing square roots in finite fields. *Information Processing Letters*, 98(1):1–5, 2006.

[LP24] Shai Levin and Robi Pedersen. *Faster Proofs and VRFs from Isogenies.* Cryptology ePrint Archive, Report 2024/1626, 2024.

[NO24] Kohei Nakagawa and Hiroshi Onuki. QFESTA: Efficient algorithms and parameters for FESTA using quaternion algebras. In Reyzin and Stebila [RS24], pages 75–106.

[OM22] Hiroshi Onuki and Tomoki Moriya. Radical isogenies on montgomery curves. In Goichiro Hanaoka, Junji Shikata, and Yohei Watanabe, editors, *PKC 2022, Part I*, volume 13177 of *LNCS*, pages 473–497. Springer, Cham, March 2022.

[RS24] Leonid Reyzin and Douglas Stebila, editors. *CRYPTO 2024, Part V*, volume 14924 of *LNCS*. Springer, Cham, August 2024.

[Sco20] Michael Scott. A note on the calculation of some functions in finite fields: Tricks of the trade. Cryptology ePrint Archive, Report 2020/1497, 2020.

[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.