Computing 2^a-isogenies in Legendre Form

Jesse Elliott[*†] Aaron Hutchinson[‡§] David Jao[§]

Abstract

We introduce a method for efficiently computing 2^a-isogenies in Legendre form with applications in post-quantum cryptography. An example of a secure application is the Charles-Goren-Lauter (CGL) hash function [3], which recently saw significant improvement in complexity by Doliskani et al. [4].

The majority of work on isogeny computation uses elliptic curves in Montgomery form; this includes the original work on SIDH by Jao, De Feo and Plût [10] and the state of the art implementation of SIKE [8]. Elliptic curves in twisted Edwards form have also been used due to their efficient elliptic curve arithmetic, and complete Edwards curves have been used for their benefit of providing added security against side channel attacks (see Azarderakhsh et al. [1]). As far as we know, elliptic curves in Legendre form have not yet been explored for isogeny-based cryptography. Legendre form has the benefit of a very simple defining equation, and the simplest possible representation of the 2-torsion subgroup. In this work, we develop a new framework for constructing 2^a-isogenies using elliptic curves in Legendre form, and in doing so optimize Legendre curve arithmetic and 2-isogeny computations on Legendre curves by avoiding any square root computations.

Keywords— Post-quantum cryptography, supersingular elliptic curves, cryptographic hash functions

[*]Inria Paris & IMJ-PRG, Sorbonne Université, Paris, France

[†]David R. Cheriton School of Computer Science, University of Waterloo, Waterloo, Canada

[‡]Cyber Engineering & Mathematics and Statistics, Louisiana Tech University, Ruston, USA

[§]Department of Combinatorics & Optimization and Institute for Quantum Computing, University of Waterloo, Waterloo, Canada

1 Introduction

Isogeny-based cryptography is an active area of research in post-quantum cryptography. Isogeny-based schemes are generally compact with small key sizes, while computational time is relatively slower compared to other post-quantum frameworks. Several cryptosystems are built on the hypothesized computational hardness of the *supersingular isogeny path problem*: given primes p and ℓ, and two supersingular elliptic curves E and E' over $\mathbb{F}_{p^2}$, find a path from E to E' in the ℓ-isogeny graph. An *ℓ-isogeny graph* is an $(\ell+1)$-regular graph with supersingular elliptic curves as vertices and ℓ-isogenies as edges. Furthermore, over $\mathbb{F}_{p^2}$, ℓ-isogeny graphs are *Ramanujan*, expander graphs with non-trivial eigenvalues having absolute value at most $2\sqrt{\ell}$. Consequently the graph is highly connected and random walks on the graph have rapid mixing: by starting at a vertex and randomly moving from one vertex to another, a state is quickly reached where the probability of being at any given vertex is almost the same, regardless of the starting point. Therefore the distribution of the walk's position approaches uniform. Recovering the path is hypothesized to be computationally hard. In particular, the pre-image resistance of the Charles-Goren-Lauter (CGL) hash function [3] is based on the hardness of the ℓ-isogeny path problem. We introduce a method for efficiently computing 2^a-isogenies in Legendre form. An example of a secure application in post-quantum cryptography is the CGL hash function [3], which recently underwent a significant improvement in complexity by Doliskani et al. [4]. As far as we know, elliptic curves in Legendre form have not yet been explored for isogeny-based cryptography. Legendre form has the benefit of a very simple defining equation, and the simplest possible representation of the 2-torsion subgroup. We develop a new framework for constructing 2^a-isogenies using elliptic curves in Legendre form, and in doing so optimize Legendre curve arithmetic and 2-isogeny computations on Legendre curves by avoiding any square root computations.

This work first appeared in the author's PhD thesis [5].

1.1 Background

Elliptic Curves and Isogenies. Let $\mathbb{F}_q$ be a finite field with q elements, where $q = p^k$ for some prime p. Let E_1 and E_2 be elliptic curves over $\mathbb{F}_q$. An *isogeny* $\varphi : E_1 \to E_2$ is a non-constant rational map preserving the identity element, and also a group homomorphism from $E_1(\mathbb{F}_q)$ to $E_2(\mathbb{F}_q)$. The *degree* of an isogeny φ, denoted $\deg \varphi$, is its degree as a rational map. When an isogeny is separable and non-constant, then its degree is finite and equal to the size of its kernel. An isogeny of degree ℓ is called an *ℓ-isogeny*. (See [11, 13] for more background on elliptic curves and isogenies.) Moreover, non-constant separable isogenies are determined by their kernel, up to isomorphism, in the following sense.

Theorem 1. ([11, Proposition 4.12]) Let E be an elliptic curve and let G be a finite subgroup of E. There exists a unique (up to isomorphism) elliptic curve E' and a separable isogeny $\varphi : E \to E'$ with $\ker(\varphi) = G$.

The curve E' in the theorem above is often labeled E/G and is called the *quotient of E by G*. Vélu's formulas (see Vélu [12]) provide the equations for the codomain curve E/G and for the isogeny $\varphi : E \to E/G$. For a point P in E, Vélu's formulas give explicit equations $\varphi(P) = (\varphi_1(P), \varphi_2(P))$

where

$$\varphi_1(P) = x(P) + \sum_{Q \in G \setminus \{\mathcal{O}\}} [x(P+Q) - x(Q)]$$
$$\varphi_2(P) = y(P) + \sum_{Q \in G \setminus \{\mathcal{O}\}} [y(P+Q) - y(Q)].$$

Furthermore, non-constant separable isogenies can be factored into a composition of isogenies of prime degree over $\mathbb{F}_q$. In particular, if φ is a non-constant and separable isogeny with degree ℓ^e for some prime ℓ, then φ can be factored into a composition $\varphi = \varphi_e \varphi_{e-1} \cdots \varphi_1$ where $\deg \varphi_i = \ell$ for all $1 \leq i \leq e$.

For an elliptic curve E defined over $\mathbb{F}_q$ and an integer n, the multiplication by n map $[n] : E \to E$ is an isogeny. The kernel of $[n]$ is the n-torsion subgroup $E[n]$ of E, given by $E[n] = \{P \in E \ : \ [n]P = \mathcal{O}\}$. If n does not divide p, then $E[n]$ is isomorphic to $\mathbb{Z}/n\mathbb{Z} \oplus \mathbb{Z}/n\mathbb{Z}$ as an abelian group.

We say that E_1 and E_2 are *isogenous* if there exists a non-constant isogeny $\varphi : E_1 \to E_2$. For any such φ, there always exists a *dual isogeny* $\hat{\varphi} : E_2 \to E_1$ satisfying $\deg \varphi = \deg \hat{\varphi}$ and $\varphi\hat{\varphi} = \hat{\varphi}\varphi = [\deg \varphi]$, making the property of being isogenous an equivalence relation.

An *endomorphism* of an elliptic curve E defined over $\mathbb{F}_q$ is an isogeny $\varphi : E \to E$. The set of all endomorphisms, together with point-wise addition and function composition, forms the structure of a ring called the *endomorphism ring*, which is denoted by $\mathrm{End}(E)$. We say that E is *supersingular* when $\dim_{\mathbb{Z}}(\mathrm{End}(E)) = 4$ and we say that E is *ordinary* when $\dim_{\mathbb{Z}}(\mathrm{End}(E)) = 2$. Isogenous elliptic curves are either both supersingular or both ordinary. Every supersingular elliptic curve is isomorphic to an elliptic curve that is defined over a finite field of order p^2. The work in this paper focuses exclusively on supersingular elliptic curves, and therefore we assume in the remainder that $q = p^2$.

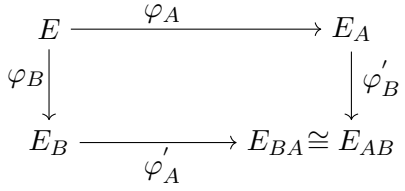

Figure 1: A high-level depiction of Supersingular Isogeny Diffie-Hellman. The initial curve E is given as a public parameter. In round 1, Alice (resp. Bob) constructs the secret isogeny φ_A (resp. φ_B). In round 2, Alice (resp. Bob) constructs another secret isogeny φ'_A (resp. φ'_B). The shared secret is the j-invariant of the final curve $E_{AB} \cong E_{BA}$.

Supersingular Isogeny Diffie-Hellman. SIDH is a key exchange algorithm proposed by Jao and De Feo [9] that was conjectured to be post-quantum secure before the attack in 2022 by Castryck and Decru [2]. Our isogeny computation builds upon the SIDH framework. We reference SIDH throughout the paper and employ the same terminology. Note that our use of the SIDH framework does not introduce security risks, since we only use the framework for isogeny construction.

The security of SIDH is not based on the supersingular isogeny path problem but rather the *computational supersingular isogeny problem*: let $\varphi : E \to E_R$ be an isogeny whose kernel is

$\langle P_R + \gamma Q_R \rangle$, where γ is chosen at random from $\mathbb{Z}/l_R^e\mathbb{Z}$ with $l_R \in \{2,3\}$ and $\{P_R, Q_R\}$ generators of $E[l_R^e]$. Given E_R and the values $\varphi(P_S), \varphi(Q_S)$, for generators $\{P_S, Q_S\}$ of $E[l_S^f]$ with $l_S \in \{2,3\}$ and $l_S \neq l_R$, find a generator R of $\langle P_R + \gamma Q_R \rangle$. The additional torsion data are exploited in the attack by Castryck and Descru [2].

We now describe SIDH at a high level. Let $p = 2^a 3^b f \pm 1$ be a prime number, where f is a small co-factor and $2^a \approx 3^b$. Let E be a supersingular elliptic curve defined over $\mathbb{F}_{p^2}$.

The key exchange provided by SIDH is a variation of Diffie-Hellman. The public parameters consist of a supersingular elliptic curve E and points $P_A, Q_A, P_B, Q_B \in E$ such that $E[2^a] = \langle P_A, Q_A \rangle$ and $E[3^b] = \langle P_B, Q_B \rangle$. In the *first round* of SIDH, Alice chooses integers $m_A, n_A \in \mathbb{Z}/2^a\mathbb{Z}$ as her secret key and computes an isogeny $\varphi_A : E \to E_A$ with

$$E_A := E/\langle m_A P_A + n_A Q_A \rangle, \qquad \ker(\varphi_A) = \langle m_A P_A + n_A Q_A \rangle.$$

Similarly, Bob chooses integers $m_B, n_B \in \mathbb{Z}/3^b\mathbb{Z}$ as his secret key and computes an isogeny $\varphi_B : E \to E_B$ with

$$E_B := E/\langle m_B P_B + n_B Q_B \rangle, \qquad \ker(\varphi_B) = \langle m_B P_B + n_B Q_B \rangle.$$

Now, akin to Diffie-Hellman, Alice and Bob exchange some information: Alice sends $(E_A, \varphi_A(P_B), \varphi_A(Q_B))$ to Bob, and Bob sends $(E_B, \varphi_B(P_A), \varphi_B(Q_A))$ to Alice. The *second round* then begins, where Alice computes an isogeny $\varphi'_A : E_B \to E_{BA}$ and Bob computes an isogeny $\varphi'_B : E_A \to E_{AB}$ such that:

$$E_{BA} := E_B/\langle m_A \varphi_B(P_A) + n_A \varphi_B(Q_A) \rangle, \qquad E_{AB} := E_A/\langle m_B \varphi_A(P_B) + n_B \varphi_A(Q_B) \rangle,$$
$$\ker(\varphi'_A) = \langle m_A \varphi_B(P_A) + n_A \varphi_B(Q_A) \rangle, \qquad \ker(\varphi'_B) = \langle m_B \varphi_A(P_B) + n_B \varphi_A(Q_B) \rangle.$$

The two curves E_{AB} and E_{BA} are isomorphic since they are both isomorphic to the curve

$$E/\langle m_B P_B + n_B Q_B, m_A P_A + n_A Q_A \rangle.$$

The shared key is then the j invariant of $E_{BA} \cong E_{AB}$.

Supersingular Isogeny Key Encapsulation. The most current and optimized version of SIDH is Supersingular Isogeny Key Encapsulation (SIKE) [8]. In 2016, Galbraith, Petit, Shani and Ti showed that if a static public key is used in SIDH then the protocol becomes vulnerable to an *active attack*, where an advisary can fully recover the corresponding secret key in a number of iterations equalling the bit-length of the secret [6]. In SIKE, one applies a variant of the Fujisaki-Okamoto transform to force Bob to reveal his encryption key to Alice, which Alice then uses to re-encrypt Bob's ciphertext and verify its validity, thus preventing the active attack.

Computing Isogenies in SIDH: Strategies. In both rounds of SIDH, both Alice and Bob are tasked with computing (and possibly evaluating) an isogeny $\varphi : E \to E/\langle R \rangle$ for some curve E and some point $R \in E$ of order ℓ^e for $\ell \in \{2,3\}$ and $e \in \{a, b\}$, such that $\ker(\varphi) = \langle R \rangle$. Applying Vélu's formulas directly to $\langle R \rangle$ to construct φ is not efficient; the computation requires $O(\deg \varphi) = O(\ell^e) = O(\sqrt{p})$ field operations. For better efficiency, the isogeny construction is instead broken up into a decomposition of prime degree isogenies φ_i, $1 \leq i \leq e$, with $\varphi = \varphi_e \varphi_{e-1} \cdots \varphi_1$. Using this method, φ can be computed with $e \times O(\deg \varphi_i) = O(e\ell)$ field operations. We now describe in detail how the construction of $\varphi_e, \ldots, \varphi_1$ is made efficient.

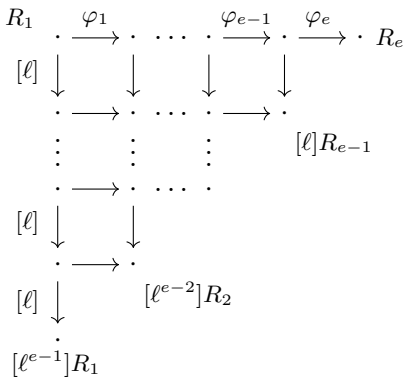

Figure 2: A visual representation of the computational structure of computing $\varphi = \varphi_e \dots \varphi_1$.

As in Jao and De Feo. [10], we let $E_1 = E$ and $R_1 = R$, and for $1 \leq i \leq e$ define

$$E_{i+1} = E_i/\langle \ell^{e-i} R_i \rangle, \quad \varphi_i : E_i \to E_{i+1}, \quad R_{i+1} = \varphi_i(R_i).$$

Then $E/\langle R \rangle = E_e$ and $\varphi = \varphi_e \dots \varphi_1$ has kernel $\langle R \rangle$. Each curve E_{i+1} and isogeny φ_i (which has degree ℓ) can be computed using Vélu's formulas once the subgroup $\langle \ell^{e-i} R_i \rangle$ of E_i is known.

Figure 2 provides a visual representation of the computational structure of the problem. Vertices represent elliptic curve points, with points on the same positive diagonal having the same order, and points on the same vertical belonging to the same curve. Downward edges represent point multiplications by ℓ, and right edges represent ℓ-isogeny evaluations. The goal is to compute all the order ℓ points along the bottom diagonal, from which we can apply Vélu's formulas to compute the corresponding φ_i. Figure 2 motivates the idea of a strategy, originally defined in Jao De Feo and J. Plût [10], and the following three definitions are taken from [7].

Definition 1. For positive integers n, let $T_n = (V, E)$ be the graph defined as follows:

1. The set of vertices V consists of all points in the plane with integer coordinates which lie inside or on the boundary of the region bounded by the lines $x = 0, y = 0$, and $y = x - n + 1$.
2. The edge set E consists of all line segments of unit length which connect two vertices in V.

It follows from Definition 1 that every edge of T_n is either horizontal or vertical. We turn T_n into a directed graph by orienting all horizontal edges to the right and all vertical edges downward. We say that a vertex is a *leaf* if it has no outgoing edges (consequently, the leaves of T_n are exactly the integral points on the line $y = x - n + 1$).

Definition 2. A **strategy** S is a subgraph of T_n such that:

1. The vertex $(0, 0)$ and all vertices on the line $y = x - n + 1$ are in S.
2. For each vertex v on the line $y = x - n + 1$, there is a path from $(0, 0)$ to v in S.
3. No two edges of S share the same target.
4. There are no leaves in S distinct from the leaves of T_n.

We write $|S| = n$ when S is a strategy in T_n. Any strategy yields a valid algorithm to compute the isogeny $\varphi = \varphi_e \dots \varphi_1$ by decorating T_e as in Figure 2 (a consequence of Lemma 4.2 from Jao et al. [10]).

We now give a method for combining two strategies together to form a larger strategy. For strategies S_1 and S_2 with $|S_1| = n_1$ and $|S_2| = n_2$, define a strategy $S_1 \# S_2$ in $T_{n_1+n_2}$ by:

1. $S_1 \# S_2$ contains the (unique) path connecting $(0, 0)$ to $(n_2, 0)$.
2. $S_1 \# S_2$ contains the (unique) path connecting $(0, 0)$ to $(0, -n_1)$.
3. $S_1 \# S_2$ contains S_1 as a subgraph, shifted to the right n_2 units.
4. $S_1 \# S_2$ contains S_2 as a subgraph, shifted down n_1 units.

We refer to the binary operation # as *join*; note that it is nonassociative and noncommutative.

Definition 3. A strategy S in T_n is **canonical** if S can be expressed as $n - 1$ many applications of the join operator on the strategy T_1. That is, S is some parenthesization of $\underbrace{T_1 \# T_1 \# \dots \# T_1}_{\text{n}}$.

Two naive canonical strategies immediately come to mind. We define the nth *multiplication-based* strategy A_n and *isogeny-based* strategy B_n recursively as $A_1 = B_1 = T_1$ and $A_{n+1} = T_1 \# A_n$ and $B_{n+1} = B_n \# T_1$. Figure 3 shows 3 strategies in T_5: the multiplication-based strategy, the isogeny-based strategy, and a canonical strategy.

When a strategy is used in SIDH, horizontal edges correspond to isogeny evaluations and vertical edges correspond to point multiplications. Furthermore, all vertical edges have a common computational cost and all horizontal edges have a common computational cost. To get a sense of the computational effort required to construct an isogeny of degree ℓ^e with a given strategy, we assign weights to each edge in a strategy. We define a *measure* as a pair $(\mathfrak{p}, \mathfrak{q})$ of positive real numbers, and turn a strategy S into a weighted strategy as follows.

Definition 4. A **weighted strategy** with the measure $(\mathfrak{p}, \mathfrak{q})$ is a strategy turned into a weighted graph, where vertical edges have weight $\mathfrak{p}$ and horizontal edges have weight $\mathfrak{q}$. The **cost** of the weighted strategy is the sum of the weights of all the edges in the strategy. An **optimal strategy** is one of minimal cost.

Proposition 4.6 of Jao et al. [10], shows that every optimal strategy must be canonical.

Montgomery Curves. A Montgomery curve $M_{\mathfrak{b},\mathfrak{a}}$ is given by an equation of the form $\mathfrak{b}y^2 = x^3 + \mathfrak{a}x^2 + x$, where $\mathfrak{a}, \mathfrak{b} \in \mathbb{F}_q$ are the curve coefficients. In isogeny-based cryptography, most formulas do not require the value of $\mathfrak{b}$ and it is often left unspecified. To avoid field inversions, it is often useful to projectivize the variable $\mathfrak{a}$ as $[\mathfrak{a} : 1] \in \mathbb{P}^1$, so that $[A : C] = [\mathfrak{a} : 1]$ if and only if $C\mathfrak{a} = A$. Furthermore, the most efficient implementations involving Montgomery curves, such as [8], use the quantity $[A_{24}^+ : C_{24}]$ in place of $[A : C]$, where $[A_{24}^+ : C_{24}] = [A + 2C : 4C]$.

The Choice of Curve Model. The majority of work based on an SIDH framework uses elliptic curves in Montgomery form; this includes the original work by Jao, De Feo and Plût [9, 10] and the state of the art, SIKE [8]. Elliptic curves in twisted Edwards form have also been used because

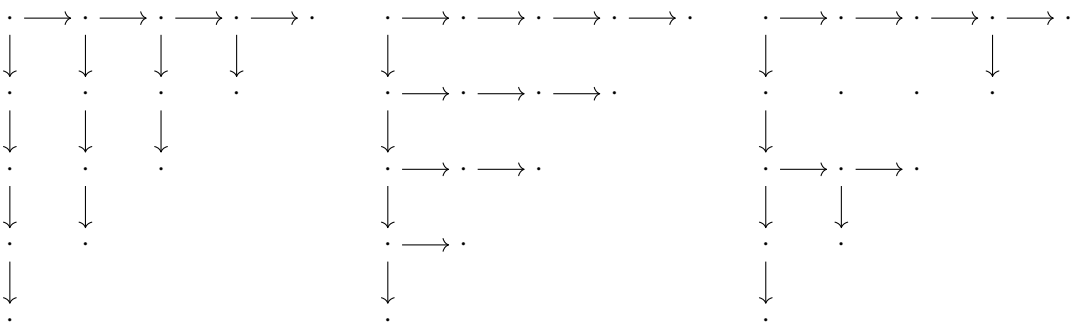

Figure 3: The multiplication-based strategy (left), the isogeny-based strategy (middle), and a canonical strategy (right) in T_5.

of their efficient elliptic curve arithmetic, and complete Edwards curves have been used for their benefit of providing added security against side channel attacks (see Azarderakhsh et al. [1]).

As far as we know, elliptic curves in Legendre form have not yet been explored for isogeny-based cryptography. Legendre form has the benefit of a very simple defining equation, and the simplest possible representation of the 2-torsion subgroup. We therefore find motivation in using Legendre form for constructing 2^a-isogenies. Our algorithms are founded on the SIDH framework and find applications for instance in isogeny computation for the CGL hash function [3, 4].

1.2 Contributions

The main contributions of this work are as follows.

1. We describe in detail how a degree 2^a isogeny can be constructed between elliptic curves in Legendre form for use in isogeny-based cryptographic protocols such as the CGL hash function [3, 4]. To achieve this, we derive many results for elliptic curves in Legendre form and degree 2 isogenies between curves in Legendre form in Section 2. Naively constructing a degree 2 isogeny between Legendre curves requires the computation of a square root. Our formula for 2-isogenies shows that the computation of this square root can be avoided when an order 4 point above the kernel generator is known. Furthermore, we also describe how to convert between Montgomery form and Legendre form and give explicit isomorphisms for this task.

2. We detail algorithms which implement the theoretic results on Legendre curves from Section 2. In particular, we describe Algorithm 7, which can be viewed as an analogue of Algorithm 19 of SIKE [8] adapted to the setting of Legendre curves. Algorithm 7 takes as input the projectivized coefficients $[\kappa_1 : \mu_1]$ of a Legendre curve $L_{[\kappa_1:\mu_1]}$ and the coordinates $(X_R : Z_R)$ of a point on $L_{[\kappa_1:\mu_1]}$ of order exactly 2^a. The algorithm constructs a degree 2^a isogeny $\varphi : L_{[\kappa_1:\mu_1]} \to L_{[\kappa_{a+1}:\mu_{a+1}]}$ to some Legendre curve $L_{[\kappa_{a+1}:\mu_{a+1}]}$ such that $\ker(\varphi) = \langle R \rangle$; this is performed by constructing a sequence of 2-isogenies $\varphi_1, \ldots, \varphi_a$, such that $\varphi_i : L_{[\kappa_i:\mu_i]} \to L_{[\kappa_{i+1}:\mu_{i+1}]}$ is a degree 2 isogeny between Legendre curves $L_{[\kappa_i:\mu_i]}$ and $L_{[\kappa_{i+1}:\mu_{i+1}]}$, for $1 \leq i \leq a$, with $\varphi = \varphi_a \cdots \varphi_1$. Algorithm 7 relies on Algorithms 1, 2, 3, and 4 which efficiently implement the results from Section 2.

3. In Section 4 we further describe an open problem—the original motivation for this work—which if solved would skip the strategy traversal performed in SIDH altogether, using the fact

that the 2-torsion of Legendre curves has a simple representation and is known in advance for all intermediate curves.

1.3 Paper Organization

In Section 2, we discuss elliptic curves in Legendre form. We give many results regarding their arithmetic, torsion structure, and construction of 2-isogenies between them. We also describe how to convert between Montgomery form and Legendre form and give explicit isomorphisms for this task. In Section 3, we provide a collection of algorithms which implement the results from Section 2; these are designed for 2^a-isogeny computation with applications for instance to the CGL hash function [3, 4]. We discuss these algorithms in detail and explain how they relate to the results from Section 2. In Section 4, we give concluding remarks and discuss some future work. In Appendix A, we provide pseudocode for the algorithms.

2 Formulas for Legendre Form

This section provides formulas for various operations on elliptic curves in Legendre form. Throughout we use $\mathbb{F}$ to denote an arbitrary field.

2.1 Legendre Form

For $\lambda \in \mathbb{F}$ with $\lambda \neq 0, 1$, the *Legendre curve* L_λ *with coefficient* λ is the elliptic curve in $\mathbb{P}^2(\mathbb{F})$ given by the affine equation

$$L_\lambda : y^2 = x(x-1)(x-\lambda), \tag{1}$$

where λ is called the *Legendre coefficient*. The *j-invariant* of L_λ is the value

$$j(\lambda) = 2^8 \frac{(\lambda^2 - \lambda + 1)^3}{\lambda^2(\lambda-1)^2} \in \mathbb{F},$$

and two curves are isomorphic if and only if they have the same j-invariant. The mapping $j : \mathbb{F} \setminus \{0, 1\} \to \mathbb{F}$ given by $\lambda \mapsto j(\lambda)$ is exactly six-to-one, except above $j(\lambda) = 0$ and $j(\lambda) = 1728$; the six values mapping to $j(\lambda) \neq 0, 1728$ under j are

$$\left\{\lambda, \frac{1}{\lambda}, 1-\lambda, \frac{1}{1-\lambda}, \frac{\lambda}{\lambda-1}, \frac{\lambda-1}{\lambda}\right\},$$

and therefore these six values define the same Legendre curve up to isomorphism.

2.2 Projective Coordinates and Coefficients

We embed affine $\mathbb{F}$ space into projective space $\mathbb{P}(\mathbb{F})$ using the usual mapping. Specifically, $\lambda \in \mathbb{F}$ is represented in projective form as $[\lambda : 1]$. Projective form has the property that for any $\kappa, \mu \in \mathbb{F}$ not both zero and $c \in \mathbb{F}$ nonzero, we have that $[\kappa : \mu] = [c\kappa : c\mu]$. We will sometimes abuse notation by writing $\lambda = [\lambda : 1]$, and so $[\kappa : \mu] = \kappa/\mu$ when μ is nonzero.

For efficiency reasons, it will be useful to use projective coordinates for the coefficient of our elliptic curves. We therefore give an alternative definition of a Legendre curve which uses a curve coefficient in projective form.

Definition 5. Let $[\kappa : \mu] \in \mathbb{P}(\mathbb{F})$ with $(\kappa - \mu)\kappa\mu \neq 0$. The *Legendre curve* $L_{[\kappa:\mu]}$ *with coefficient* $[\kappa : \mu]$ is the elliptic curve in $\mathbb{P}^2(\overline{\mathbb{F}})$ defined by the homogeneous polynomial

$$L_{[\kappa:\mu]} : \quad \mu Y^2 Z = X(X - Z)(\mu X - \kappa Z). \tag{2}$$

Note that the curve is well defined even when changing the representation of $[\kappa : \mu]$. The Legendre curve $L_{[\kappa:\mu]}$ is identical to the Legendre curve L_λ from the previous section, where $\lambda = \kappa/\mu$. In particular, the j-invariant of $L_{[\kappa:\mu]}$ is

$$j(\kappa/\mu) = \frac{256(\kappa^2 - \kappa\mu + \mu^2)^3}{\mu^2\kappa^2(\kappa - \mu)^2}.$$

Using the projective form of the curve coefficient will help reduce the computational cost of constructing isogenies later in this section. Going forward, we work entirely with projective coefficients.

2.3 Legendre Arithmetic

Points on the Legendre curve $L_{[\kappa:\mu]}$ are elements of $\mathbb{P}^2(\overline{\mathbb{F}})$ which satisfy the defining homogeneous polynomial given in Equation 2. These points form an abelian group through the *group law*, with the single *point at infinity* $\mathcal{O} = [0 : 1 : 0]$ playing the role of the group identity. The following theorem describes part of the group law for Legendre form.

Theorem 2. Let $L_{[\kappa:\mu]}$ be a Legendre curve. The points of $L_{[\kappa:\mu]}$ form an abelian group with identity $\mathcal{O} = [0 : 1 : 0]$, and the doubling of $[X : Y : Z] \in L_{[\kappa:\mu]}$ is given by $2[X : Y : Z] = [X' : Y' : Z']$, where

$$\begin{aligned} X' &= (\mu X^2 - \kappa Z^2)^2, \\ Y' &= \frac{(\mu X^2 - 2\kappa XZ + \kappa Z^2)(\mu X^2 - 2\mu XZ + \kappa Z^2)(\mu X^2 - \kappa Z^2)}{2\mu YZ}, \\ Z' &= 4\mu XZ(\mu X^2 - (\kappa + \mu)XZ + \kappa Z^2) \end{aligned}$$

when $YZ \neq 0$.

Proof. Legendre form is a special case of generalized Weierstrass form, and so the theorem follows immediately from well-known results. See Silverman [11] for details. □

Notice that X' and Z' above depend only on X, Z, κ, and μ, and so even if the value of Y is not known repeated doubling may be partially computed by ignoring Y'. If the X and Z coordinates of some point are known, the value of Y can be determined up to sign through Equation 2.

2.4 The 2-Torsion of a Legendre Curve

One of the most appealing features of a Legendre curve is the simplicity of their 2-torsion points. Recall for $n \in \mathbb{N}$ that the *n-torsion* of a curve $L_{[\kappa:\mu]}$ is the subgroup $L_{[\kappa:\mu]}[n] = \{P \in L_{[\kappa:\mu]} : nP = \mathcal{O}\}$.

Theorem 3. The 2-torsion subgroup of $L_{[\kappa:\mu]}$ is isomorphic to $\mathbb{Z}/2\mathbb{Z} \times \mathbb{Z}/2\mathbb{Z}$ and has the form

$$L_{[\kappa:\mu]}[2] = \{\mathcal{O}, [0 : 0 : 1], [1 : 0 : 1], [\kappa : 0 : \mu]\}.$$

Given $[X : Y : Z] \in L_{[\kappa:\mu]}[2]$, at first glance the nonuniqueness of projective representations of points may give the impression that determining which of the four above points $[X : Y : Z]$ corresponds to may require at least a field multiplication. Upon closer inspection we see the following algorithm, requiring at most three field equality checks:

$$[X : Y : Z] = \begin{cases} \mathcal{O} & \text{if } Z = 0, \\ [0 : 0 : 1] & \text{if } X = 0, \\ [1 : 0 : 1] & \text{if } X = Z, \\ [\kappa : 0 : \mu] & \text{otherwise.} \end{cases}$$

2.5 The 4-Torsion of a Legendre Curve

Here we give a description of $L_{[\kappa:\mu]}[4]$. The 4-torsion consists of 16 points, 4 of which form the 2-torsion subgroup $L_{[\kappa:\mu]}[2]$ given previously. Aside from these, there are 12 points of order 4, which can be partitioned into 3 subsets based upon which 2-torsion point they double to. That is, if $T \in L_{[\kappa:\mu]}[2]$ has order 2, then each of the subsets $S_T = \{P \in L_{[\kappa:\mu]} : 2P = T\}$ has size 4 and they do not intersect each other. For such a fixed T, the points within S_T can be written to have a very similar expression, which only differs by at most two minus signs. The computation to determine each S_T is tedious, and the end result is given in Figure 4. The three points in the Order 4 row of Figure 4 each represent one of the sets S_T, where the value of T is the point in the Order 2 row directly below.

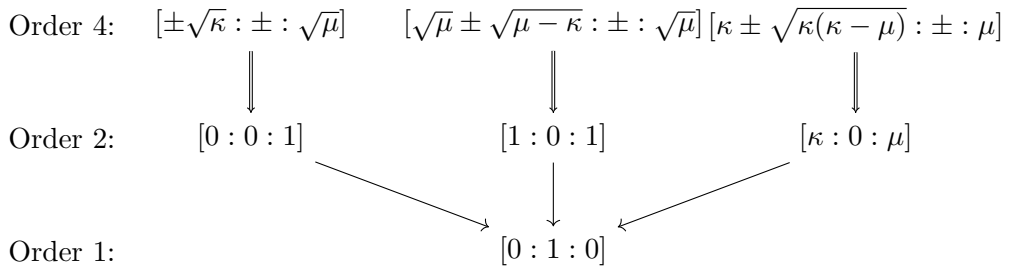

Figure 4: The 4-torsion of $L_{[\kappa:\mu]}$. Each point in the Order 4 row above actually represents four distinct points of order 4 on $L_{[\kappa:\mu]}$, found by taking all possible choices of sign; for brevity the Y-coordinates have been omitted. All arrows represent the doubling map, with double arrows indicating that all four points at the tail are mapped to the same point at the tip.

2.6 2-Isogenies: Legendre to Legendre

We will be primarily interested in 2-isogenies between Legendre curves. Such isogenies will have a kernel generated by some 2-torsion point, which were fully described in Section 2.4. This subsection gives constructions for 2-isogenies between Legendre curves, restricting to the cases in which the kernel is generated by either of the 2-torsion points $T_0 := [0 : 0 : 1]$ or $T_1 := [1 : 0 : 1]$.

One may attempt to apply Velu's formulas directly to $T \in \{[0 : 0 : 1], [1 : 0 : 1]\}$ to obtain a degree 2 isogeny $\psi : L_{[\kappa:\mu]} \to E$, and then post compose with an isomorphism which puts E into Legendre form. This approach works, except that the resulting expressions for the codomain curve coefficients and point image coordinates involve square roots of κ and μ. Since in most instances computing these square roots is slow, we avoid this problem by using the coordinates of an order 4

point P_4 such that $2P_4 = T$. Theorems 4 and 5 below give formulas for a degree 2 isogeny between Legendre curves using the coordinates of P_4.

Theorem 4. Let $L_{[\kappa:\mu]}$ be a Legendre curve. Assume that $P_4 = [X_{P_4} : Y_{P_4} : Z_{P_4}]$ is a point of order 4 on the curve with $2P_4 = [0 : 0 : 1]$. The 2-isogeny whose kernel is $\langle [0 : 0 : 1] \rangle$ can be written in the form $\varphi : L_{[\kappa:\mu]} \to L_{[\kappa':\mu']}$ with $\varphi([X : Y : Z]) = [X' : Y' : Z']$ for any $[X : Y : Z] \in L_{[\kappa:\mu]}$, where

$$\begin{aligned} \kappa' &= \kappa Z_{P_4} + 2\mu X_{P_4} + \mu Z_{P_4}, & X' &= \mu Z_{P_4} X^2 + 2\mu X_{P_4} XZ + \kappa Z_{P_4} Z^2, \\ \mu' &= 4\mu X_{P_4}, & Y' &= \frac{\mu X^2 - \kappa Z^2}{2(\kappa/\mu)^{1/4} X} Z_{P_4} Y, \\ & & Z' &= \mu' XZ. \end{aligned}$$

Proof. The theorem can be verified as follows. Write $L_{[\kappa:\mu]}$ in affine form L_λ and apply Velu's formulas for generalized Weierstrass form on L_λ with kernel $\langle (0, 0) \rangle$. This results in a 2-isogeny $\psi : L_\lambda \to E$ for some curve E. One can then construct an isomorphism $\theta : E \to L_{\lambda'}$ to a Legendre curve $L_{\lambda'}$ using well-known results (Proposition 2.16 of [13]). Projectivizing L_λ, $L_{\lambda'}$ and the composition $\theta\psi$ yields a 2-isogeny between Legendre curves with kernel $[0 : 0 : 1]$, but many of the expressions involve square roots. The expressions stated in the theorem can be obtained using the fact that $X_{P_4}\sqrt{\mu} = Z_{P_4}\sqrt{\kappa}$, where $P_4 = [X_{P_4} : Y_{P_4} : Z_{P_4}] \in L_{[\kappa:\mu]}$ satisfies $2P_4 = [0 : 0 : 1]$. □

For φ as above, notice that $\varphi([1 : 0 : 1]) = [\kappa' : 0 : \mu'] = \varphi([\kappa : 0 : \mu])$. Consequently, it follows from the theory of dual isogenies that the isogeny having kernel $\langle [\kappa' : 0 : \mu'] \rangle$ corresponds to the dual of φ.

Theorem 5. Let $L_{[\kappa:\mu]}$ be a Legendre curve. Assume that $P_4 = [X_{P_4} : Y_{P_4} : Z_{P_4}]$ is a point of order 4 on the curve with $2P_4 = [1 : 0 : 1]$. The 2-isogeny whose kernel is $\langle [1 : 0 : 1] \rangle$ can be written in the form $\varphi : L_{[\kappa:\mu]} \to L_{[\kappa':\mu']}$ with $\varphi([X : Y : Z]) = [X' : Y' : Z']$ for any $[X : Y : Z] \in L_{[\kappa:\mu]}$, where

$$\begin{aligned} \kappa' &= \kappa Z_{P_4} + 2\mu X_{P_4} - 4\mu Z_{P_4}, & X' &= \mu Z_{P_4} X^2 + (\kappa' - \kappa Z_{P_4}) XZ - \kappa' Z^2, \\ \mu' &= 4\mu X_{P_4} - 4\mu Z_{P_4}, & Y' &= \frac{\mu X^2 - 2\mu XZ + \kappa Z^2}{2(1 - \frac{\kappa}{\mu})^{1/4}(X - Z)} Z_{P_4} Y, \\ & & Z' &= \mu'(XZ - Z^2). \end{aligned}$$

The proof for Theorem 5 is very similar to that of Theorem 4. As before, for φ as in Theorem 5 we have that $\varphi([0 : 0 : 1]) = [\kappa' : 0 : \mu'] = \varphi([\kappa : 0 : \mu])$, and so $\langle [\kappa' : 0 : \mu'] \rangle$ is the subgroup corresponding to the dual of φ.

Because $[\kappa' : 0 : \mu'] \in L_{[\kappa:\mu]}$ always corresponds to the dual of φ when $\ker(\varphi) = \langle T \rangle$ for $T \in \{[0 : 0 : 1], [1 : 0 : 1]\}$, isogenies with kernel $\langle [\kappa : 0 : \mu] \rangle$ never need to be considered when constructing a degree 2^a isogeny with kernel $\langle R \rangle$ such that $2^{a-1}R \neq [\kappa : 0 : \mu]$. This fact will be used implicitly when formulating algorithms for 2^a-isogeny computation (see Algorithm 7, to come).

2.7 2-Isogenies: Legendre to Montgomery

The following theorem describes how to construct a 2-isogeny from a Legendre curve to a Montgomery curve, and also how to obtain the j-invariant of the codomain curve.

Theorem 6. Let $L_{[\kappa:\mu]}$ be a Legendre curve. Define the curves M_0 and M_1 and maps $\varphi_0 : L_{[\kappa:\mu]} \to M_0$ and $\varphi_1 : L_{[\kappa:\mu]} \to M_1$ by:

$$M_0 : \frac{\mu}{\kappa-\mu}Y^2Z = X^3 + \frac{2(\kappa+\mu)}{\kappa-\mu}X^2Z + XZ^2$$
$$M_1 : \quad \frac{\mu}{\kappa}Y^2Z = X^3 + \frac{2(\kappa-2\mu)}{\kappa}X^2Z + XZ^2$$
$$\varphi_0([X:Y:Z]) = [(X-Z)(\mu X - \kappa Z) : \frac{\mu X^2 - \kappa Z^2}{X}Y : (\kappa-\mu)XZ]$$
$$\varphi_1([X:Y:Z]) = [X(\mu X - \kappa Z) : \mu(X - Z + \frac{(\kappa-\mu)Z^2}{(X-Z)\mu})Y : \kappa(X-Z)Z].$$

Then M_0 and M_1 are Montgomery curves and φ_0 and φ_1 are degree 2 isogenies with $\ker(\varphi_0) = \langle [0:0:1] \rangle$ and $\ker(\varphi_1) = \langle [1:0:1] \rangle$. The j-invariants of M_0 and M_1 are

$$j(M_0) = \frac{(\kappa^2 + 14\kappa\mu + \mu^2)^3}{(\kappa\mu(\kappa-\mu))^2}, \qquad j(M_1) = \frac{(\kappa^2 - 16\kappa\mu + 16\mu^2)^3}{(\kappa\mu(\kappa-\mu))^2}.$$

Proof. Everything in the theorem can be verified as follows. We work in affine coordinates so that $\kappa = \lambda$ and $\mu = 1$. Apply the standard Velu's formulas for Weierstrauss form to L_λ in each case to arrive at the (affinized) curves

$$E_0 : y^2 = x^3 - (\lambda+1)x^2 - 4\lambda x + 4\lambda(\lambda+1),$$
$$E_1 : y^2 = x^3 - (\lambda+1)x^2 + (6\lambda-5)x - (\lambda-1)(4\lambda-3).$$

The curves E_0 and E_1 contain the 2-torsion points $P_0 := (\lambda+1, 0)$ and $P_1 := (\lambda-1, 0)$, respectively. The other 2-torsion points on each curve can then be found by factoring the cubic defining the curve. The curves M_0 and M_1 (respectively isomorphic to E_0 and E_1) can be derived by using an affine change of coordinates to some curve $By^2 = x^3 + Ax^2 + x$ such that P_i is sent to $(0,0)$ and the x-coordinates of the other 2-torsion points are sent to the zeros of $x^2 + Ax + 1$. Composing the Velu maps with these change of coordinates and projectivizing the result yields φ_0 and φ_1. □

Note of course the j-invariant is unaffected by what form the curve is in – Montgomery or Legendre. Therefore Theorem 6 can be used to compute the j-invariant in the last step of Algorithm 4.

2.8 Isomorphisms: Montgomery and Legendre

Here we state formulas for converting between Legendre and Montgomery forms of elliptic curves.

Lemma 1. Let $M_{A,B}$ be a Montgomery curve. Define $\kappa = A^2 - A\sqrt{A^2-4} - 2$ and $\mu = 2$. Then

$$\varphi_{M,L} : M_{A,B} \to L_{[\kappa:\mu]}, \qquad \varphi_{M,L}([X:Y:Z]) = \left[2X : \sqrt{\frac{8B}{-A-\sqrt{A^2-4}}}Y : -(A+\sqrt{A^2-4})Z\right]$$
$$\varphi_{L,M} : L_{[\kappa:\mu]} \to M_{A,B}, \qquad \varphi_{L,M}([X:Y:Z]) = \left[-(A+\sqrt{A^2-4})X : \frac{(-A-\sqrt{A^2-4})^{3/2}}{\sqrt{8B}}Y : 2Z\right]$$

are inverse isomorphisms.

Proof. The Lemma follows from well-known results. See Silverman 2008 [11] and Washington [13] 2008 for details. □

If one instead starts with a Legendre curve $L_{[\kappa:\mu]}$ and defines $A = (\kappa + \mu)/\sqrt{\kappa\mu}$ and $B = 1$, the same maps above allow one to convert from Legendre to Montgomery form.

3 Algorithms

Here we discuss our algorithms that implement the results given in previous sections.

3.1 The Main Algorithm

Algorithm 7 is the top-level algorithm that computes the 2^a-isogeny $\varphi : L_{[\kappa:\mu]} \to L_{[\kappa':\mu']}$ with $\ker(\varphi) = \langle R \rangle$ for a point R of exact order 2^a with $2^{a-1}R \neq [\kappa : 0 : \mu]$. Algorithm 7 can be seen as the analog of Algorithm 19 from SIKE [8] adapted to our setting. However, note that Algorithm 7 uses 2-isogenies whereas Algorithm 19 of SIKE [8] uses 4-isogenies. Furthermore, since points of order 4 are used as a means of constructing 2-isogenies, we use a strategy S in T_{a-1}. The strategy S is computed ahead of time using a dynamic programming algorithm (see D. Jao et al. [10, Section 4.2.2] for details), and is given as a parameter to Algorithm 7 in linearized form. The input of Algorithm 7 consists of the Legendre curve coefficients $[\kappa : \mu] \in \mathbb{P}^1$ and the coordinates $[X_R : Z_R]$ of the point $R \in L_{[\kappa:\mu]}$ with the Y coordinate omitted. The j-invariant of the final curve $L_{[\kappa':\mu']}$ is given as output.

3.2 Subroutines

By traversing the strategy S, Algorithm 7 constructs a many 2-isogenies of the form $\varphi_i : L_{[\kappa_i:\mu_i]} \to L_{[\kappa_{i+1}:\mu_{i+1}]}$ for $1 \leq i \leq a$. The composition $\varphi_a \dots \varphi_1$ is the desired isogeny φ from Section 3.1. To ease the presentation, Algorithm 7 uses Algorithms 1, 2, 3 and 4 as subroutines, which we give an overview of now. Throughout all algorithms, the Y coordinate of every point is omitted.

Algorithm 1 computes the codomain curve coefficients $[\kappa_{i+1} : \mu_{i+1}]$ for an isogeny of the form $\varphi_i : L_{[\kappa_i:\mu_i]} \to L_{[\kappa_{i+1}:\mu_{i+1}]}$, as well as coefficients c_0, c_1, c_2 used to evaluate φ_i at a point. Algorithm 1 takes the coefficients $[\kappa_i : \mu_i]$ and a point $P_4 \in L_{[\kappa_i:\mu_i]}$ of order (exactly) 4 as input, where it is assumed that $2P_4 \in \{[0 : 0 : 1], [1 : 0 : 1]\}$. First, the algorithm checks whether $2P_4 = [0 : 0 : 1]$ is true by checking the value of $\mu X_{P_4}^2 - \kappa Z_{P_4}^2$, the partial doubling of P_4 (see Theorem 2); a bit b is set as 0 if $\mu X_{P_4}^2 = \kappa Z_{P_4}^2$ and 1 otherwise, where $b = 0$ indicates that $2P_4 = [0 : 0 : 1]$. Depending on the value of b, the values of $[\kappa_{i+1} : \mu_{i+1}]$ and c_0, c_1, c_2 are computed according to Theorem 4 or Theorem 5. Note that $\bar{b} = 1$ if and only if $b = 0$. The quantities c_0, c_1, c_2 are the coefficients of X^2, XZ, Z^2, respectively, in the expression for X' of Theorem 4 or Theorem 5. Algorithm 1 uses conditional swaps to avoid branching based on the value of $2P_4$.*

Algorithm 2 evaluates the 2-isogenies $\varphi_i : L_{[\kappa_i:\mu_i]} \to L_{[\kappa_{i+1}:\mu_{i+1}]}$ at a given point. It receives the codomain curve coefficient μ_{i+1} (κ_{i+1} is not needed), the evaluation coefficients c_0, c_1, c_2 and the bit variable b computed in Algorithm 1, and a point $Q \in L_{[\kappa_i:\mu_i]}$ to evaluate under φ_i. The output of Algorithm 2 is the coordinates of the point $\varphi(Q) \in L_{[\kappa_{i+1}:\mu_{i+1}]}$, computed using the formulas given in Theorems 4 and 5. Again, conditional swaps are used with the value of b to avoid branching.

*Note that we use the command cswap(α, β, b), which swaps the values of α and β conditioned on $b = 1$.

Algorithm 3 performs coordinate doubling for Legendre curves. The input consists of curve coefficients $[\kappa : \mu]$ for a Legendre curve $L_{[\kappa:\mu]}$ and the coordinates $(X_P : Z_P)$ for a point $P \in L_{[\kappa:\mu]}$. The output consists of the coordinates $(X_{[2]P} : Z_{[2]P})$ corresponding to $[2]P$, computed using Theorem 2.

Algorithm 4 computes the j-invariant of the final Legendre curve $L_{[\kappa_{a+1}:\mu_{a+1}]}$ from the isogeny φ_{a-1} and the point $P_4 \in L_{[\kappa_{a-1}:\mu_{a-1}]}$ used to generate φ_{a-1}. The input to Algorithm 4 is the same as that of Algorithm 1. Similar to Algorithm 1, Algorithm 4 performs a partial evaluation of the input point P_4 to determine if the image is $[0 : 0 : 1]$ on $L_{[\kappa_a:\mu_a]}$. Depending on the result, the proper formula from Theorem 6 is used to compute the j-invariant of a Montogmery curve $M_{\mathfrak{b},\mathfrak{a}}$ using conditional swaps; this curve is isomorphic to the Legendre curve $L_{[\kappa_{a+1}:\mu_{a+1}]}$ therefore having the same j-invariant. In this way, the curve coefficients of $L_{[\kappa_{a+1}:\mu_{a+1}]}$ and the isogeny φ_a coefficients are not computed.

3.3 Montgomery algorithms

Algorithm 5 computes the quantities $[A_{24}^+ : C_{24}]$ of the codomain Montgomery curve $M_{\mathfrak{b},\mathfrak{a}}$ of an isogeny $\varphi' : L_{[\kappa':\mu']} \to M_{\mathfrak{b},\mathfrak{a}}$, as well as an isogeny evaluation coefficient C. The input to Algorithm 5 consists of the curve coefficients $[\kappa' : \mu']$, a point $(X_{P_4} : Z_{P_4}) \in L_{[\kappa:\mu]}$ of exact order 4 on the curve $L_{[\kappa:\mu]}$, and the evaluation coefficients c_0, c_1, c_2 for the isogeny $\varphi : L_{[\kappa:\mu]} \to L_{[\kappa':\mu']}$ with $\ker(\varphi) = \langle 2P_4 \rangle$. The point P_4 here can be seen as lying in the upper right corner of the strategy S (see Section 1). Since $\ker(\varphi) = \langle 2P_4 \rangle$, we have that $\varphi(P_4) \in \{[0 : 0 : 1], [1 : 0 : 1]\}$. Algorithm 5 partially evaluates P_4 under φ and checks whether $X_{\varphi(P_4)}$ is zero. A bit variable b' is assigned 0 if $X_{\varphi(P_4)}$ is zero and 1 otherwise. The values of $[A_{24}^+ : C_{24}]$ are then computed according to Theorem 6, where $b = 0$ indicates using the formula for M_0 and $b = 1$ indicates using that of M_1. The evaluation coefficient C is also computed, whose value is either $\kappa' - \mu'$ when $b = 0$ or κ' when $b = 1$. Again, conditional swaps are used based on b.

Algorithm 6 evaluates $\varphi' : L_{[\kappa':\mu']} \to M_{\mathfrak{b},\mathfrak{a}}$ at a given point. It receives as input the coordinates $(X_Q : Z_Q)$ of a point $Q \in L_{[\kappa':\mu']}$, the quantity C from Algorithm 5 for evaluating φ', and the bit variable b' for which $\ker(\varphi') = \langle [b' : 0 : 1] \rangle$. The output is the point $(X_{\varphi'(Q)} : Z_{\varphi'(Q)})$, computed according to Theorem 6.

3.4 Algorithm Costs

Table 1 summarizes the computational costs of the subroutines Algorithms 1–6. The costs are given in terms of **m**, **s**, and **i**, which denote the costs of multiplication, squaring, and inversion in $\mathbb{F}_{p^2}$. We assume the costs of addition, negation, flipping bits, and the cswap function are negligible.

4 Conclusion

Future Work. Here we expand on our initial motivation for using Legendre curves for isogeny-based cryptography. As explained in Section 1, at a low level the SIDH framework constructs an ℓ^e-isogeny φ as a sequence of ℓ-isogenies $\varphi_i : E_i \to E_{i+1}$ with $\ker(\varphi_i) = \langle R_i \rangle$. The points R_i are obtained through the laborious task of strategy traversal and are only needed in order to construct φ_i. When $\ell = 2$, each R_i is a nontrivial 2-torsion point on the curve E_i. If E_i is a Legendre curve L_{λ_i}, then $L_{\lambda_i}[2] = \{\mathcal{O}, (0, 0), (1, 0), (\lambda, 0)\}$ and so R_i has a particularly simple form.

Alg.	Function	Cost
1	`2IsoLegCurve`	$5\mathbf{m}$
2	`2IsoLegEval`	$4\mathbf{m} + 3\mathbf{s}$
3	`coordinateDBL`	$5\mathbf{m} + 4\mathbf{s}$
4	`jInv`	$8\mathbf{m} + 6\mathbf{s} + \mathbf{i}$
5	`2IsoMontCurve`	$4\mathbf{m} + 2\mathbf{s}$
6	`2IsogMontEval`	$5\mathbf{m}$

Table 1: Computational costs of each algorithm in terms of field arithmetic costs. Here, **m**, **s**, and **i** denote the costs of multiplication, squaring, and inversion in $\mathbb{F}_{p^2}$, respectively. The costs of addition and negation are considered to be negligible and are not reported in the cost values.

The appeal for using Legendre curves is that since the 2-torsion is known in advance and is identical for all curves, one may consider an alternative approach for constructing the isogeny φ. If each φ_i is constructed so that R_i is chosen as $(0,0)$ or $(1,0)$, Section 2 showed that $(\lambda, 0)$ will correspond to the dual isogeny; in other words, the value of each R_i corresponds to some bit β_i, where $R_i = (\beta_i, 0)$. In this way, one could use some bitstring $(\beta_1, \ldots, \beta_e)$ and *choose* to use the value of $R_i = (\beta_i, 0)$ for each kernel generator rather than derive R_i through strategy traversal. This would have the effect of skipping strategy traversal altogether—improving efficiency by a $O(\log d)$ factor where d is the tree depth—since all points R_i are known from the start.

The drawback to this approach (as shown in Section 2) is that while applying Velu's formulas to L_{λ_i} and $(\beta_i, 0)$ is easy and results in an efficiently computable isogeny $\varphi_i : L_{\lambda_i} \to E_{i+1}$, it is rarely the case that E_{i+1} is also in Legendre form and so this efficiency is lost when attempting to make the approach iterable. In particular, the isomorphism which puts E_{i+1} in Legendre form requires computing a square root, which is relatively costly. It is true that for sufficient tree depth d, avoiding the strategy traversal would provide improvements in efficiency even after accounting for the square root computation. The main difficulty is that, for isogenies of Vélu type, the order of the kernel generator (that is, the degree of the isogeny) is always less than the group order, which is typically not much greater than p, and under these conditions the tree depth will never be large enough that trading off strategy traversal for square root computations saves time. Consequently, Legendre form would outperform the strategy traversal approach only if we can find a way to avoid the square root computation.

Concluding Remarks. In this work, we detailed many results regarding Legendre curves, including point doubling formulas, the 4-torsion structure, and formulas for 2-isogenies between Legendre curves. Furthermore, we used these theoretical results to devise algorithms that construct 2^a-isogenies between Legendre curves. The work has applications in isogeny-based cryptography where 2^a-isogeny computation is required, in particular the CGL hash function [3, 4].

References

[1] R. Azarderakhsh, E. Bakos Lang, D. Jao, and B. Koziel. Edsidh: Supersingular isogeny diffie-hellman key exchange on edwards curves. In Anupam Chattopadhyay, Chester Rebeiro, and Yuval Yarom, editors, *Security, Privacy, and Applied Cryptography Engineering*, pages 125–141. Springer, 2018.

[2] Wouter Castryck and Thomas Decru. An efficient key recovery attack on SIDH. In Carmit Hazay and Martijn Stam, editors, *Advances in Cryptology – EUROCRYPT 2023*, pages 423–447. Springer, 2023.

[3] D. X. Charles, E. Z. Goren, and K. E. Lauter. Cryptographic hash functions from expander graphs. *Journal of Cryptology*, 22:93–113, 2009.

[4] Javad Doliskani, Geovandro C. C. F. Pereira, and Paulo S. L. M. Barreto. Faster cryptographic hash function from supersingular isogeny graphs. In *Selected Areas in Cryptography – SAC 2022*, pages 399–415. Springer, 2022.

[5] Jesse Elliott. *On the Complexity of Feasibility, Connectivity, and 2^a-Isogeny Computation.* PhD thesis, University of Waterloo, 2025.

[6] Steven D. Galbraith, Christophe Petit, Barak Shani, and Yan Bo Ti. On the security of supersingular isogeny cryptosystems. In *Advances in Cryptology – ASIACRYPT 2016*, pages 63–91. Springer, 2016.

[7] Aaron Hutchinson, Jason LeGrow, Brian Koziel, and Reza Azarderakhsh. Further optimizations of CSIDH: A systematic approach to efficient strategies, permutations, and bound vectors. In Mauro Conti, Jianying Zhou, Emiliano Casalicchio, and Angelo Spognardi, editors, *Applied Cryptography and Network Security*, pages 481–501. Springer, 2020.

[8] D. Jao, R. Azarderakhsh, M. Campagna, C. Costello, L. De Feo, B. Hess, A. Hutchinson, A. Jalali, K. Karabina, B. Koziel, B. LaMacchia, P. Longa, M. Naehrig, G. Pereira, J. Renes, V. Soukharev, and D. Urbanik. Supersingular isogeny key encapsulation, 2020. Available at https://sike.org/files/SIDH-spec.pdf.

[9] D. Jao and L. De Feo. Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies. In Bo-Yin Yang, editor, *Post-Quantum Cryptography*, pages 19–34. Springer, 2011.

[10] D. Jao, L. De Feo, and J. Plût. Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies. *Journal of Mathematical Cryptology*, 8(3):209–247, 2014.

[11] Joseph H. Silverman. *The Arithmetic of Elliptic Curves.* Graduate Texts in Mathematics. Springer, 2 edition, 2008.

[12] J. Vélu. Isogénies entre courbes elliptiques. *C. R. Acad. Sci. Paris, Sér. A*, 273:238–241, 1971.

[13] Lawrence C. Washington. *Elliptic Curves: Number Theory and Cryptography.* CRC Press, 2008.

A Pseudocode

Algorithm 1: Computing a 2-isogenous Legendre curve

Function: `2IsoLegCurve`

Input: *Curve coefficients* $(\kappa : \mu)$ *defining a curve* $L_{[\kappa:\mu]}$*;a point* $(X_{P_4} : Z_{P_4}) \in L_{[\kappa:\mu]}$ *with exact order* 4.

Output: *Curve coefficients* $(\kappa' : \mu')$ *defining a curve* $L_{[\kappa':\mu']}$*; evaluation coefficients* c_0, c_1, c_2 *for an isogeny* $\varphi : L_{[\kappa:\mu]} \to L_{[\kappa':\mu']}$*; a bit variable* b *satisfying* $\ker(\varphi) = \langle [b : 0 : 1] \rangle$.

$t_0 \leftarrow \mu \cdot X_{P_4}$

$t_1 \leftarrow \kappa \cdot Z_{P_4}$

$t_2 \leftarrow \mu \cdot Z_{P_4}$

$s_0 \leftarrow t_0 \cdot X_{P_4}$ // $s_0 = \mu X_{P_4}^2$

$s_1 \leftarrow t_1 \cdot Z_{P_4}$ // $s_1 = \kappa Z_{P_4}^2$

$b \leftarrow s_0 \overset{?}{\neq} s_1$ // $b = 0$ `(Theorem 4)`; $b = 1$ `(Theorem 5)`

// κ'

$r_0 \leftarrow -t_2$ // $r_0 = -\mu Z_{P_4}$

$r_0 \leftarrow r_0 + r_0$ // $r_0 = -2\mu Z_{P_4}$

$r_0 \leftarrow r_0 + r_0$ // $r_0 = -4\mu Z_{P_4}$

$\mathrm{cswap}(r_0, t_2, \overline{b})$ // $b = 0$: $r_0 = \mu Z_{P_4}$; $b = 1$: $r_0 = -4\mu Z_{P_4}$

$\kappa' \leftarrow r_0$ // $b = 0$: $\kappa' = \mu Z_{P_4}$; $b = 1$: $\kappa' = -4\mu Z_{P_4}$

$\kappa' \leftarrow \kappa' + t_1$ // $b = 0$: $\kappa' = \kappa Z_{P_4} + \mu Z_{P_4}$; $b = 1$: $\kappa' = \kappa Z_{P_4} - 4\mu Z_{P_4}$

$r_1 \leftarrow t_0 + t_0$ // $r_1 = 2\mu X_{P_4}$

$\kappa' \leftarrow \kappa' + r_1$ // $b = 0$: $\kappa' = \kappa Z_{P_4} + 2\mu X_{P_4} + \mu Z_{P_4}$; $b = 1$: $\kappa' = \kappa Z_{P_4} + 2\mu X_{P_4} - 4\mu Z_{P_4}$

// μ'

$r_2 \leftarrow t_0$ // $r_2 = \mu X_{P_4}$

$r_2 \leftarrow t_0 + t_0$ // $r_2 = 2\mu X_{P_4}$

$r_2 \leftarrow t_0 + t_0$ // $r_2 = 4\mu X_{P_4}$

$\mathrm{cswap}(r_0, 0, \overline{b})$ // $b = 0$: $r_0 = 0$; $b = 1$: $r_0 = -4\mu Z_{P_4}$

$r_3 \leftarrow r_2 + r_0$ // $b = 0$: $r_3 = 4\mu X_{P_4}$; $b = 1$: $r_3 = 4\mu X_{P_4} - 4\mu Z_{P_4}$

$\mu' \leftarrow r_3$ // $b = 0$: $\mu' = 4\mu X_{P_4}$; $b = 1$: $\mu' = 4\mu X_{P_4} - 4\mu Z_{P_4}$

// c_0, c_1, c_2 `(coefficients for` X'`)`

$c_0 \leftarrow t_2$ // $c_0 = \mu Z_{P_4}$

$r_4 \leftarrow -t_1$ // $r_4 = -\kappa Z_{P_4}$

$r_4 \leftarrow r_4 + \kappa'$ // $r_4 = \kappa' - \kappa Z_{P_4}$

$r_2 \leftarrow t_0$ // $r_2 = \mu X_{P_4}$

$r_2 \leftarrow r_2 + t_0$ // $r_2 = 2\mu X_{P_4}$

$\mathrm{cswap}(r_4, r_2, \overline{b})$ // $b = 0$: $r_4 = 2\mu X_{P_4}$; $b = 1$: $r_4 = \kappa' - \kappa Z_{P_4}$

$c_1 \leftarrow r_4$ // $b = 0$: $c_1 = 2\mu X_{P_4}$; $b = 1$: $c_1 = \kappa' - \kappa Z_{P_4}$

$r_5 \leftarrow -\kappa'$ // $r_5 = -\kappa'$

$\mathrm{cswap}(r_5, t_1, \overline{b})$ // $b = 0$: $r_5 = \kappa Z_{P_4}$; $b = 1$: $r_5 = -\kappa'$

$c_2 \leftarrow r_5$ // $b = 0$: $c_2 = \kappa Z_{P_4}$; $b = 1$: $c_2 = -\kappa'$

Return $(\kappa' : \mu'), c_0, c_1, c_2, b$

Algorithm 2: Evaluating a 2-isogenous Legendre curve at a point

Function: `2IsoLegEval`

Input: *A point* $(X_Q : Z_Q)$ *on some curve* $L_{[\kappa:\mu]}$*; evaluation coefficients* c_0, c_1, c_2 *for an isogeny* $\varphi : L_{[\kappa:\mu]} \to L_{[\kappa':\mu']}$*; the codomain curve coefficient* μ'*; a bit variable* b *satisfying* $\ker(\varphi) = \langle [b : 0 : 1] \rangle$.

Output: *A point* $(X_{Q'} : Z_{Q'})$*, where* $\varphi(Q) = Q'$.

$t_0 \leftarrow X_Q^2$

$t_1 \leftarrow (X_Q + Z_Q)^2$ // $t_1 = X_Q^2 + 2X_QZ_Q + Z_Q^2$

$t_2 \leftarrow Z_Q^2$

$t_1 \leftarrow t_1 - t_0$ // $t_1 = 2X_QZ_Q + Z_Q^2$

$t_1 \leftarrow t_1 - t_2$ // $t_1 = 2X_QZ_Q$

$t_1 \leftarrow t_1/2$ // $t_1 = X_QZ_Q$

// $X_{Q'}$

$X_{Q'} \leftarrow c_0 \cdot t_0$ // $X_{Q'} = c_0X_Q^2$

$s_0 \leftarrow c_1 \cdot t_1$ // $s_0 = c_1X_QZ_Q$

$X_{Q'} \leftarrow X_{Q'} + s_0$ // $X' = c_0X_Q^2 + c_1X_QZ_Q$

$s_1 \leftarrow c_2 \cdot t_2$ // $s_1 = c_2Z_Q^2$

$X_{Q'} \leftarrow X_{Q'} + s_1$ // $X_{Q'} = c_0X_Q^2 + c_1X_QZ_Q + c_2Z_Q^2$

// $Z_{Q'}$

$r_0 \leftarrow t_1$ // $r_0 = X_QZ_Q$

$r_0 \leftarrow r_0 - t_2$ // $r_0 = X_QZ_Q - Z_Q^2$

$\mathrm{cswap}(r_0, t_1, \bar{b})$ // $b = 0$: $r_0 = X_QZ_Q$; $b = 1$: $r_0 = X_QZ_Q - Z_Q^2$

$Z_{Q'} \leftarrow \mu' \cdot r_0$ // $b = 0$: $Z_{Q'} = \mu' X_QZ_Q$; $b = 1$: $Z_{Q'} = \mu'(X_QZ_Q - Z_Q^2)$

Return $(X_{Q'} : Z_{Q'})$

Algorithm 3: Coordinate doubling

Function: `coordinateDBL`

Input: *Curve coefficients* $(\kappa : \mu)$ *defining a curve* $L_{[\kappa:\mu]}$*; a point* $(X_P : Z_P) \in L_{[\kappa:\mu]}$.

Output: *A point* $(X_{[2]P} : Z_{[2]P})$*, the doubling of* P.

1	$t_0 \leftarrow X_P^2$	
2	$t_1 \leftarrow \mu \cdot t_0$	// $t_1 = \mu X_P^2$
3	$t_2 \leftarrow Z_P^2$	
4	$t_3 \leftarrow \kappa \cdot t_2$	// $t_3 = \kappa Z_P^2$
		// $X_{[2]P}$
5	$X_{[2]P} \leftarrow t_1 - t_3$	// $X_{[2]P} = \mu X_P^2 - \kappa Z_P^2$
6	$X_{[2]P} \leftarrow X_{[2]P}^2$	// $X_{[2]P} = (\mu X_P^2 - \kappa Z_P^2)^2$
		// $Z_{[2]P}$
7	$r_0 \leftarrow X_P + Z_P$	
8	$r_0 \leftarrow r_0^2$	// $r_0 = (X_P + Z_P)^2 = X_P^2 + Z_P^2 + 2X_P Z_P$
9	$r_0 \leftarrow r_0 - t_0$	// $r_0 = Z_P^2 + 2X_P Z_P$
10	$r_0 \leftarrow r_0 - t_2$	// $r_0 = 2X_P Z_P$
11	$r_0 \leftarrow r_0/2$	// $r_0 = X_P Z_P$
12	$r_1 \leftarrow \kappa + \mu$	
13	$r_1 \leftarrow r_0 \cdot r_1$	// $r_1 = (\kappa + \mu) X_P Z_P$
14	$Z_{[2]P} \leftarrow t_1 - r_1$	// $Z_{[2]P} = \mu X_P^2 - (\kappa + \mu) X_P Z_P$
15	$Z_{[2]P} \leftarrow Z_{[2]P} + t_3$	// $Z_{[2]P} = \mu X_P^2 - (\kappa + \mu) X_P Z_P + \kappa Z_P^2$
16	$Z_{[2]P} \leftarrow r_0 \cdot Z_{[2]P}$	// $Z_{[2]P} = X_P Z_P (\mu X_P^2 - (\kappa + \mu) X_P Z_P + \kappa Z_P^2)$
17	$Z_{[2]P} \leftarrow \mu \cdot Z_{[2]P}$	// $Z_{[2]P} = \mu X_P Z_P (\mu X_P^2 - (\kappa + \mu) X_P Z_P + \kappa Z_P^2)$
18	$Z_{[2]P} \leftarrow Z_{[2]P} + Z_{[2]P}$	// $Z_{[2]P} = 2\mu X_P Z_P (\mu X_P^2 - (\kappa + \mu) X_P Z_P + \kappa Z_P^2)$
19	$Z_{[2]P} \leftarrow Z_{[2]P} + Z_{[2]P}$	// $Z_{[2]P} = 4\mu X_P Z_P (\mu X_P^2 - (\kappa + \mu) X_P Z_P + \kappa Z_P^2)$
20	**Return** $(X_{[2]P} : Z_{[2]P})$	

Algorithm 4: j-invariant computation

Function: `jInv`

Input: *Curve coefficients* $(\kappa' : \mu') \in \mathbb{P}^1$*; a point* $(X_{P_4} : Z_{P_4}) \in L_{[\kappa:\mu]}$ *where* P_4 *has exact order* 4 *on some curve* $L_{[\kappa:\mu]}$*; evaluation coefficients* c_0, c_1, c_2 *for an isogeny* $\varphi : L_{[\kappa:\mu]} \to L_{[\kappa':\mu']}$ *with* $\ker(\varphi) = \langle 2P_4 \rangle$.

Output: *The* j*-invariant* $j(L_{[\kappa'':\mu'']})$ *where* $L_{[\kappa'':\mu'']}$ *is the codomain of the isogeny* $\varphi' : L_{[\kappa':\mu']} \to L_{[\kappa'':\mu'']}$ *whose kernel is generated by* P_4 *with* $\ker(\varphi') = \langle \varphi(P_4) \rangle$.

$t_0 \leftarrow X_{P_4}^2$

$t_1 \leftarrow X_{P_4} \cdot Z_{P_4}$

$t_2 \leftarrow Z_{P_4}^2$

// X'_{P_4}

$X_{P_4'} \leftarrow c_0 \cdot t_0$ // $X_{P_4'} = c_0 X_{P_4}^2$

$s_0 \leftarrow c_1 \cdot t_1$ // $s_0 = c_1 X_{P_4} Z_{P_4}$

$X_{P_4'} \leftarrow X_{P_4'} + s_0$ // $X' = c_0 X_{P_4}^2 + c_1 X_{P_4} Z_{P_4}$

$s_1 \leftarrow c_2 \cdot t_2$ // $s_1 = c_2 Z_{P_4}^2$

$X_{P_4'} \leftarrow X_{P_4'} + s_1$ // $X_{P_4'} = c_0 X_{P_4}^2 + c_1 X_{P_4} Z_{P_4} + c_2 Z_{P_4}^2$

$b' \leftarrow X_{P_4'} \overset{?}{\neq} 0$ // $b' = 0$ `if` $X_{P_4'} = 0$; $b' = 1$ `otherwise`

// j-invariant `numerator`

$t_0 \leftarrow \kappa'$

$t_1 \leftarrow t_0^2$ // $t_1 = \kappa'^2$

$t_2 \leftarrow \mu'$

$t_3 \leftarrow t_2^2$ // $t_3 = \mu'^2$

$s_0 \leftarrow t_3$ // $s_0 = \mu'^2$

$s_0 \leftarrow s_0 + s_0$ // $s_0 = 2\mu'^2$

$s_0 \leftarrow s_0 + s_0$ // $s_0 = 4\mu'^2$

$s_0 \leftarrow s_0 + s_0$ // $s_0 = 8\mu'^2$

$s_0 \leftarrow s_0 + s_0$ // $s_0 = 16\mu'^2$

$\mathrm{cswap}(t_3, s_0, b')$ // $b' = 0$: $t_3 = \mu'^2$; $b' = 1$: $t_3 = 16\mu'^2$

$r_0 \leftarrow \kappa \cdot \mu'$

$r_1 \leftarrow r_0 + r_0$ // $r_1 = 2\kappa'\mu'$

$r_1 \leftarrow r_1 + r_1$ // $r_1 = 4\kappa'\mu'$

$r_1 \leftarrow r_1 + r_1$ // $r_1 = 8\kappa'\mu'$

$r_1 \leftarrow r_1 + r_1$ // $r_1 = 16\kappa'\mu'$

$r_2 \leftarrow -r_1$ // $r_2 = -16\kappa'\mu'$

$r_3 \leftarrow r_1 - r_0$ // $r_3 = 15\kappa'\mu'$

$r_3 \leftarrow r_3 - r_0$ // $r_3 = 14\kappa'\mu'$

$\mathrm{cswap}(r_3, r_2, b')$ // $b' = 0$: $r_3 = 14\kappa'\mu'$; $b' = 1$: $r_3 = -16\kappa'\mu'$

$N \leftarrow t_1$ // $N = \kappa^2$

$N \leftarrow N + r_3$ // $b' = 0$: $N = \kappa'^2 + 14\kappa'\mu'$; $b' = 1$: $N = \kappa'^2 - 16\kappa'\mu'$

$N \leftarrow N + t_3$ // $b' = 0$: $N = \kappa'^2 + 14\kappa'\mu' + \mu'^2$; $b' = 1$: $N = \kappa'^2 - 16\kappa'\mu' + 16\mu'^2$

// j-invariant `denominator`

$D_0 \leftarrow r_0$ // $D_0 = \kappa'\mu'$

$D_1 \leftarrow \kappa' - \mu'$ $D_0 \leftarrow D_0 \cdot D_1$ // $D_0 = \kappa'\mu'(\kappa' - \mu')$

$D_0 \leftarrow D_0^2$ // $D_0 = (\kappa'\mu'(\kappa' - \mu'))^2$

$j \leftarrow 1/D_0$ // $j = \frac{1}{(\kappa'\mu'(\kappa'-\mu'))^2}$

$j \leftarrow j \cdot N$ // $b' = 0$: $j = \frac{(\kappa'^2+14\kappa'\mu'+\mu'^2)^3}{(\kappa'\mu'(\kappa'-\mu'))^2}$; $b' = 1$: $j = \frac{(\kappa'^2-16\kappa'\mu'+16\mu'^2)^3}{(\kappa'\mu'(\kappa'-\mu'))^2}$

Return j

Algorithm 5: Computing a 2-isogenous Montgomery curve from a Legendre curve

Function: `2IsoMontCurve`

Input: *Curve coefficients* $(\kappa' : \mu') \in \mathbb{P}^1$*; a point* $(X_{P_4} : Z_{P_4}) \in L_{[\kappa:\mu]}$ *where* P_4 *has exact order* 4 *on some curve* $L_{[\kappa:\mu]}$*; evaluation coefficients* c_0, c_1, c_2 *for an isogeny* $\varphi : L_{[\kappa:\mu]} \to L_{[\kappa':\mu']}$ *with* $\ker(\varphi) = \langle 2P_4 \rangle$.

Output: *Curve coefficients* $[A_{24}^+ : C_{24}]$ *for a Montgomery curve* $M_{\mathfrak{b},\mathfrak{a}}$ *satisfying* $4A_{24}^+ = C_{24}(A+2)$*; an isogeny evaluation coefficient* C *for an isogeny* $\varphi' : L_{[\kappa':\mu']} \to M_{\mathfrak{b},\mathfrak{a}}$ *with* $\ker(\varphi') = \langle \varphi(P_4) \rangle$*; a bit variable* b' *satisfying* $\ker(\varphi') = \langle [b' : 0 : 1] \rangle$.

1 $t_0 \leftarrow X_{P_4}^2$
2 $t_1 \leftarrow X_{P_4} \cdot Z_{P_4}$
3 $t_2 \leftarrow Z_{P_4}^2$

// X'_{P_4}

4 $X_{P_4'} \leftarrow c_0 \cdot t_0$ // $X_{P_4'} = c_0 X_{P_4}^2$
5 $s_0 \leftarrow c_1 \cdot t_1$ // $s_0 = c_1 X_{P_4} Z_{P_4}$
6 $X_{P_4'} \leftarrow X_{P_4'} + s_0$ // $X' = c_0 X_{P_4}^2 + c_1 X_{P_4} Z_{P_4}$
7 $s_1 \leftarrow c_2 \cdot t_2$ // $s_1 = c_2 Z_{P_4}^2$
8 $X_{P_4'} \leftarrow X_{P_4'} + s_1$ // $X_{P_4'} = c_0 X_{P_4}^2 + c_1 X_{P_4} Z_{P_4} + c_2 Z_{P_4}^2$
9 $b' \leftarrow X_{P_4'} \overset{?}{\neq} 0$ // $b' = 0$ `if` $X_{P_4'} = 0$; $b' = 1$ `otherwise`

// C

10 $t_0 \leftarrow \kappa'$
11 $C \leftarrow t_0 - \mu'$ // $C = \kappa' - \mu'$
12 $\mathrm{cswap}(C, t_0, b')$ // $b' = 0$: $C = \kappa' - \mu'$; $b' = 1$: $C = \kappa'$

// A

13 $s_0 \leftarrow \kappa'$
14 $s_1 \leftarrow \mu'$
15 $s_2 \leftarrow -\mu'$
16 $s_2 \leftarrow s_2 + s_2$ // $s_2 = -2\mu'$
17 $\mathrm{cswap}(s_1, s_2, b')$ // $b' = 0$: $s_1 = \mu'$; $b' = 1$: $s_1 = -2\mu'$
18 $A' \leftarrow s_1$ // $b' = 0$: $A' = \mu'$; $b' = 1$: $A' = -2\mu'$
19 $A' \leftarrow A' + \kappa'$ // $b' = 0$: $A' = \kappa' + \mu'$; $b' = 1$: $A' = \kappa' - 2\mu'$
20 $A' \leftarrow A' + A'$ // $b' = 0$: $A' = 2(\kappa' + \mu')$; $b' = 1$: $A' = 2(\kappa' - 2\mu')$

// A_{24}^+

21 $A_{24}^+ \leftarrow C$ // $A_{24}^+ = C$
22 $A_{24}^+ \leftarrow A_{24}^+ + C$ // $A_{24}^+ = 2C$
23 $A_{24}^+ \leftarrow A_{24}^+ + A$ // $A_{24}^+ = A + 2C$

// C_{24}

24 $C_{24} \leftarrow C$ // $C_{24} = C$
25 $C_{24} \leftarrow C_{24} + C_{24}$ // $C_{24} = 2C$
26 $C_{24} \leftarrow C_{24} + C_{24}$ // $C_{24} = 4C$
27 **Return** A_{24}^+, C_{24}, C, b'

Algorithm 6: Evaluating $\varphi' : L_{[\kappa':\mu']} \to M_{\mathfrak{b},\mathfrak{a}}$ at a point

Function: `2IsogMontEval`

Input: *Curve coefficients* $(\kappa' : \mu')$*; an evaluation coefficient* C *for an isogeny* $\varphi' : L_{[\kappa':\mu']} \to M_{\mathfrak{b},\mathfrak{a}}$*; a point* $Q = (X_Q : Z_Q) \in L_{[\kappa':\mu']}$ *to be evaluated; a bit variable* b' *satisfying* $\ker(\varphi') = \langle [b' : 0 : 1] \rangle$.

Output: $(X_{Q'} : Z_{Q'})$*, where* $Q' = \varphi'(Q)$.

		// $X_{Q'}$
1	$t_0 \leftarrow X_Q$	
2	$t_1 \leftarrow t_0 - Z_Q$	// $t_1 = X_Q - Z_Q$
3	$\mathrm{cswap}(t_1, t_0, b')$	// $b' = 0$: $t_1 = X_Q - Z_Q$; $b' = 1$: $t_1 = X_Q$
4	$s_0 \leftarrow \mu' \cdot X_Q$	
5	$s_1 \leftarrow \kappa' \cdot Z_Q$	
6	$s_2 \leftarrow s_0 - s_1$	// $s_2 = \mu' X_Q - \kappa' Z_Q$
7	$X_{Q'} \leftarrow t_1 \cdot s_2$	// $b' = 0$: $X_{Q'} = (X_Q - Z_Q)(\mu' X_Q - \kappa' Z_Q)$; $b' = 1$: $X_{Q'} = X_Q(\mu' X_Q - \kappa' Z_Q)$
		// $Z_{Q'}$
8	$r_0 \leftarrow X_Q$	
9	$t_1 \leftarrow r_0 - Z_Q$	// $r_1 = X_Q - Z_Q$
10	$\mathrm{cswap}(r_0, r_1, b')$	// $b' = 0$: $r_0 = X_Q$; $b' = 1$: $r_0 = X_Q - Z_Q$
11	$Z_{Q'} \leftarrow r_0$	// $b' = 0$: $Z_{Q'} = X_Q$; $b' = 1$: $Z_{Q'} = X_Q - Z_Q$
12	$Z_{Q'} \leftarrow r_0 \cdot Z_Q$	// $b' = 0$: $Z_{Q'} = X_Q Z_Q$; $b' = 1$: $Z_{Q'} = (X_Q - Z_Q)Z_Q$
13	$Z_{Q'} \leftarrow C \cdot Z_Q$	// $b' = 0$: $Z_{Q'} = C X_Q Z_Q$; $b' = 1$: $Z_{Q'} = C(X_Q - Z_Q)Z_Q$
14	**Return** $(X_{Q'} : Z_{Q'})$	

Algorithm 7: Computing and evaluating a 2^a-isogeny

Function: `2aIso`

Parameters: *Integer* $a \in \mathbb{N}$, *Strategy* $(s_1, \ldots, s_{a-2}) \in (\mathbb{N}^+)^{a-2}$

Input: *Legendre curve coefficients* $(\kappa : \mu) \in \mathbb{P}^1$*; a point* $(X_R : Z_R) \in L_{[\kappa:\mu]}$ *of exact order* 2^a *with* $2^{a-1}R \neq (\kappa : 0 : \mu)$.

Output: *The* j*-invariant of* $L_{[\kappa':\mu']}$ *such that* $\varphi : L_{[\kappa:\mu]} \to L_{[\kappa':\mu']}$ *is an isogeny with* $\ker(\varphi) = \langle R \rangle$.

Initialize empty deque $\mathbf{S}$
push$(\mathbf{S}, (a-1, (X_R : Z_R)))$
$i \leftarrow 1$
while $\mathbf{S}$ *is not empty* **do**
    $(h, (X : Z)) \leftarrow$ **pop**$(\mathbf{S})$
    if $h = 1$ **then**
        $([\kappa : \mu], (c_0, c_1, c_2), b) \leftarrow$ `2IsoLegCurve`$((\kappa : \mu), (X : Z))$ `// Alg. 1`
        Initialize empty deque $\mathbf{S}'$
        while $\mathbf{S}$ *is not empty* **do**
            $(h, (X : Z)) \leftarrow$ **pull**$(\mathbf{S})$
            $(X : Z) \leftarrow$ `2IsoLegEval`$((X : Z), (c_0, c_1, c_2), \mu, b)$ `// Alg. 2`
            push$(\mathbf{S}', (h-1, (X : Z)))$
        $\mathbf{S} \leftarrow \mathbf{S}'$
    else if $0 < s_i < h$ **then**
        push$(\mathbf{S}, (h, (X : Z)))$
        for $j = 1$ *to* s_i **do**
            $(X : Z) \leftarrow$ `coordinateDBL`$((\kappa : \mu), (X : Z))$ `// Alg. 3`
        push$(\mathbf{S}, (h - s_i, (X : Z)))$
        $i \leftarrow i + 1$
    else
        Error: invalid strategy
$jInv \leftarrow$ `jInv`$((\kappa : \mu), (X : Z), (c_0, c_1, c_2))$ `// Alg. 4`
Return $jInv$;