Fast and Frobenius: Rational Isogeny Evaluation over Finite Fields

Gustavo Banegas, Valerie Gilchrist, Anaëlle Le Dévéhat, Benjamin Smith

Fast and Frobenius: Rational Isogeny Evaluation over Finite Fields

Gustavo Banegas[1], Valerie Gilchrist[2], Anaëlle Le Dévéhat[3], Benjamin Smith[3]

[1] Qualcomm France SARL, Valbonne, France
[2] Université Libre de Bruxelles and FRIA, Brussels, Belgium
[3] Inria and Laboratoire d'Informatique de l'École polytechnique, Institut Polytechnique de Paris, Palaiseau, France

Abstract. Consider the problem of efficiently evaluating isogenies $\phi : \mathcal{E} \to \mathcal{E}/H$ of elliptic curves over a finite field $\mathbb{F}_q$, where the kernel $H = \langle G \rangle$ is a cyclic group of odd (prime) order: given $\mathcal{E}$, G, and a point (or several points) P on $\mathcal{E}$, we want to compute $\phi(P)$. This problem is at the heart of efficient implementations of group-action- and isogeny-based post-quantum cryptosystems such as CSIDH. Algorithms based on Vélu's formulæ give an efficient solution to this problem when the kernel generator G is defined over $\mathbb{F}_q$. However, for general isogenies, G is only defined over some extension $\mathbb{F}_{q^k}$, even though $\langle G \rangle$ as a whole (and thus ϕ) is defined over the base field $\mathbb{F}_q$; and the performance of Vélu-style algorithms degrades rapidly as k grows. In this article we revisit the isogeny-evaluation problem with a special focus on the case where $1 \leq k \leq 12$. We improve Vélu-style isogeny evaluation for many cases where $k = 1$ using special addition chains, and combine this with the action of Galois to give greater improvements when $k > 1$.

1 Introduction

Faced with the rising threat of quantum computing, demand for quantum-secure, or post-quantum, cryptographic protocols is increasing. Isogenies have emerged as a useful candidate for post-quantum cryptography thanks to their generally small key sizes, and the possibility of implementing post-quantum group actions which offer many simple post-quantum analogues of classical discrete-log-based algorithms (see e.g. [27]).

A major drawback of isogeny-based cryptosystems is their relatively slow performance compared with many other post-quantum systems. In this paper, we improve evaluation times for isogenies of many prime degrees $\ell > 3$ given a generator of the kernel; these computations are the fundamental building blocks

* Authors listed in alphabetical order: see `https://www.ams.org/profession/leaders/CultureStatement04.pdf`.
This work was funded in part by a FRIA grant by the National Fund for Scientific Research (F.N.R.S.) of Belgium, by the French Agence Nationale de la Recherche through ANR CIAO (ANR-19-CE48-0008), and by a *Plan France 2030* grant managed by the Agence Nationale de la Recherche (ANR-22-PETQ-0008). Date of this document: 2023-06-27.

of most isogeny-based cryptosystems. Specifically, we propose simple alternative differential addition chains to enumerate points of (subsets of) the kernel more efficiently. This speeds up many ℓ-isogeny computations over the base field by a factor depending on ℓ, and also permits a full additional factor-of-k speedup for ℓ-isogenies over $\mathbb{F}_q$ whose kernel generators are defined over an extension $\mathbb{F}_{q^k}$.

Our techniques have constructive and destructive applications. First, accelerating basic isogeny computations can speed up isogeny-based cryptosystems. The methods in §4 apply for many $\ell > 3$, so they would naturally improve the performance of commutative isogeny-based schemes such as CSIDH [5], and CSI-FiSh [4] and its derivatives (such as [12] and [14]), which require computing many ℓ-isogenies for various primes ℓ. They may also improve the performance of other schemes like SQISign [17], which computes many ℓ-isogenies in its signing process. (We discuss applications further in §6.)

In §5 we focus on rational isogenies with irrational kernels; our methods there could be used to improve the performance of Couveignes–Rostovtsev–Stolbunov key exchange (CRS) and related protocols of Stolbunov [11, 25, 28, 29], further accelerating the improvements of [16]. This is a small step forward on the road to making CRS a practical "ordinary" fallback for CSIDH in the event of new attacks making specific use of the full supersingular isogeny graph (continuing the approach of [6], for example).

Our results also have applications in cryptanalysis: the best classical and quantum attacks on commutative isogeny-based schemes involve computing massive numbers of group actions, each comprised of a large number of ℓ-isogenies (see e.g. [3] and [8]). Any algorithm that reduces the number of basic operations per ℓ-isogeny will improve the effectiveness of these attacks.

Disclaimer. In this paper, we quantify potential speedups by counting finite field operations. We make no predictions of real-world speed increases, since these depend on too many additional variables including parameter sizes; the application context; implementation choices; the runtime platform (including the specificities of the architecture, vectorization, and hardware acceleration); and the availability of optimized low-level arithmetic.

2 Background

We work over (extensions of) the base field $\mathbb{F}_q$, where q is a power of a prime $p > 3$. The symbol ℓ always denotes a prime $\neq p$. In our applications, $3 < \ell \ll p$.

Elliptic curves. For simplicity, in this work every elliptic curve will be supposed to be in a general Weierstrass form $\mathcal{E} : y^2 = f(x)$. Our algorithms and applications are focused on[4] *Montgomery models*

$$\mathcal{E} : By^2 = x(x^2 + Ax + 1) \qquad \text{where} \qquad B(A^2 - 4) \neq 0\,.$$

[4] We will focus exclusively on Montgomery models, since these are the most common in isogeny-based cryptography, but our results extend easily to other models such as traditional short Weierstrass models (for number-theoretic applications).

The multiplication-by-m map is denoted by $[m]$. The q-power Frobenius endomorphism is $\pi : (x, y) \mapsto (x^q, y^q)$.

Field operations. While the curve $\mathcal{E}$ will always be defined over $\mathbb{F}_q$, we will often work with points defined over $\mathbb{F}_{q^k}$ for $k \geq 1$. We write $\mathbf{M}$, $\mathbf{S}$, and $\mathbf{a}$ for the cost of multiplication, squaring, and adding (respectively) in $\mathbb{F}_{q^k}$. We write $\mathbf{C}$ for the cost of multiplying an element of $\mathbb{F}_{q^k}$ by an element of $\mathbb{F}_q$ (typically a curve constant, or an evaluation-point coordinate). Note that $\mathbf{C} \approx (1/k)\mathbf{M}$ (when k is not too large). Later, we will write $\mathbf{F}$ for the cost of evaluating the Frobenius map on $\mathbb{F}_{q^k}$; see §5.1 for discussion on this.

x-only arithmetic. Montgomery models are designed to optimize x-only arithmetic (see [22] and [10]). The `xADD` operation is

$$\texttt{xADD} : (x(P), x(Q), x(P-Q)) \longmapsto x(P+Q)\,;$$

it can be computed at a cost of $4\mathbf{M} + 2\mathbf{S} + 6\mathbf{a}$ using the formulæ

$$\begin{cases} X_+ = Z_- \left[(X_P - Z_P)(X_Q + Z_Q) + (X_P + Z_P)(X_Q - Z_Q)\right]^2 \,, \\ Z_+ = X_- \left[(X_P - Z_P)(X_Q + Z_Q) - (X_P + Z_P)(X_Q - Z_Q)\right]^2 \end{cases} \tag{1}$$

(where $(X_P : Z_P)$, $(X_Q : Z_Q)$, $(X_+ : Z_+)$, and $(X_- : Z_-)$ are the x-coordinates $x(P)$, $x(Q)$, $x(P+Q)$, and $x(P-Q)$, respectively.

The `xDBL` operation is

$$\texttt{xDBL} : x(P) \longmapsto x([2]P)\,;$$

it can be computed at a cost of $2\mathbf{M} + 2\mathbf{S} + \mathbf{C} + 4\mathbf{a}$ using the formulæ

$$\begin{cases} X_{[2]P} = (X_P + Z_P)^2 (X_P - Z_P)^2 \,, \\ Z_{[2]P} = (4X_P Z_P)((X_P - Z_P)^2 + ((A+2)/4)(4X_P Z_P)) \,. \end{cases} \tag{2}$$

Isogenies. Let $\mathcal{E}_1, \mathcal{E}_2$ be elliptic curves over a finite field $\mathbb{F}_q$. An isogeny $\phi : \mathcal{E}_1 \to \mathcal{E}_2$ is a non-constant morphism mapping the identity point of $\mathcal{E}_1$ to the identity point of $\mathcal{E}_2$. Such a morphism is automatically a homomorphism. For more details see [26, Chapter 3, §4]. The kernel of ϕ is a finite subgroup of $\mathcal{E}_1$, and vice versa: every finite subgroup $\mathcal{G}$ of $\mathcal{E}_1$ determines a separable *quotient isogeny* $\mathcal{E}_1 \to \mathcal{E}_1/\mathcal{G}$.

Let G be the generator of the kernel group. The kernel polynomial can be expressed as:

$$D(X) := \prod_{P \in S} (X - x(P))$$

where $S \subset \langle G \rangle$ is any subset that satisfies the conditions:

$$S \cap -S = \emptyset \qquad \text{and} \qquad S \cup -S = \langle G \rangle \setminus \{0\}\,. \tag{3}$$

Every separable isogeny $\phi : \mathcal{E}_1 \to \mathcal{E}_2$ defined over $\mathbb{F}_q$ can be represented by a rational map in the form

$$\phi : (x, y) \longmapsto \big(\phi_x(x), \phi_y(x, y)\big) \tag{4}$$

with

$$\phi_x(x) = \frac{N(x)}{D(x)^2} \qquad \text{and} \qquad \phi_y(x, y) = c \cdot y \frac{d\phi_x}{dx}(x)$$

where D is the kernel polynomial of ϕ, N is a polynomial derived from D, and c is a normalizing constant in $\mathbb{F}_q$.

Vélu's formulæ. Given a curve $\mathcal{E}$ and a finite subgroup $\mathcal{G} \subset \mathcal{E}$, Vélu [30] gives explicit formulæ for the rational functions that define a separable isogeny $\phi : \mathcal{E} \to \mathcal{E}' := \mathcal{E}/\mathcal{G}$ with kernel $\mathcal{G}$, as well as the resulting codomain curve $\mathcal{E}'$. Although the quotient curve $\mathcal{E}'$ and the isogeny ϕ are defined up to isomorphism, Vélu's formulæ construct a unique *normalized* isogeny, ensuring that if ω and ω' are the invariant differentials on $\mathcal{E}$ and $\mathcal{E}'$, respectively, then $\phi^*(\omega') = \omega$.

See Kohel's Thesis [20, §2.4] for more details about explicit isogenies and a treatment of Vélu's results better-adapted to finite fields. For more information concerning isogenies and their use in cryptography we refer the reader to [13].

3 Evaluating isogenies

Let $\mathcal{E}$ be an elliptic curve over $\mathbb{F}_q$, and let $\langle G \rangle$ be a subgroup of prime order ℓ (where ℓ is not equal to the field characteristic p). We suppose $\langle G \rangle$ is defined over $\mathbb{F}_q$; then, the quotient isogeny $\phi : \mathcal{E} \to \mathcal{E}/\langle G \rangle$ is also defined over $\mathbb{F}_q$.

When we say $\langle G \rangle$ is defined over $\mathbb{F}_q$, this means $\langle G \rangle$ is *Galois stable*: that is, $\pi(\langle G \rangle) = \langle G \rangle$ (where π is the q-power Frobenius endomorphism). We will mostly be concerned with algorithms taking $x(G)$ as an input, so it is worth noting that

$$x(G) \in \mathbb{F}_{q^{k'}} \qquad \text{where} \qquad k' := \begin{cases} k & \text{if } k \text{ is odd}\,, \\ k/2 & \text{if } k \text{ is even}\,. \end{cases}$$

The set of projective x-coordinates of the nonzero kernel points is

$$\mathcal{X}_G := \big\{(X_P : Z_P) = x(P) : P \in \langle G \rangle \setminus \{0\}\big\} \subset \mathbb{P}^1(\mathbb{F}_{q^{k'}})\,;$$

each X_P/Z_P corresponds to a root of the kernel polynomial $D(X)$, and vice versa. If $\#\langle G \rangle$ is an odd prime ℓ, then $\#\mathcal{X}_G = (\ell - 1)/2$.

3.1 The isogeny evaluation problem

We want to evaluate the isogeny $\phi : \mathcal{E} \to \mathcal{E}/\langle G \rangle$. More precisely, we want efficient solutions to the problem of Definition 1:

Definition 1 (Isogeny Evaluation). *Given an elliptic curve $\mathcal{E}$ over $\mathbb{F}_q$, a list of points $(P_1, \ldots, P_n)$ in $\mathcal{E}(\mathbb{F}_q)$, and a finite subgroup $\mathcal{G}$ of $\mathcal{E}$ corresponding to the separable isogeny $\phi : \mathcal{E} \to \mathcal{E}/\mathcal{G}$, compute $(\phi(P_1), \ldots, \phi(P_n))$.*

In most cryptographic applications, the number n of evaluation points is relatively small, especially compared to the isogeny degree ℓ. We do *not* assume the codomain curve $\mathcal{E}/\mathcal{G}$ is known. If required, an equation for the codomain curve can be interpolated through the image of well-chosen evaluation points.

For each separable isogeny ϕ of degree d defined over $\mathbb{F}_q$, there exists a sequence of primes $(\ell_1, \ldots, \ell_n)$ and a sequence of isogenies $(\phi_1, \ldots, \phi_n)$, all defined over $\mathbb{F}_q$, such that $\phi_n \circ \cdots \phi_1$ and

- $\phi_i = [\ell_i]$ (the non-cyclic case) or
- ϕ_i has cyclic kernel of order ℓ_i.

The kernel of ϕ_1 is $\ker \phi \cap \mathcal{E}[\ell_1]$, and so on. The multiplication maps $[\ell_i]$ can be computed in $O(\log \ell_i)$ $\mathbb{F}_q$-operations, so we reduce quickly to the case where ϕ has prime degree ℓ, assuming the factorization of d is known (which is always the case in our applications).

In general, the isogeny evaluation problem can be reduced to evaluating the map $\alpha \mapsto D(\alpha)$, where D is the kernel polynomial and α is in $\mathbb{F}_q$ or some $\mathbb{F}_q$-algebra (see e.g. [2, §4]). We note that the polynomial D does *not* need to be explicitly computed itself.

3.2 The Costello-Hisil algorithm

The Costello-Hisil algorithm [9] is the state-of-the-art for evaluating isogenies.[5] This algorithm is a variation of Vélu's formulæ working entirely on the level of x-coordinates, using the fact that for an ℓ-isogeny ϕ of Montgomery models with kernel $\langle G \rangle$, the rational map on x-coordinates is

$$\phi_x(x) = x \cdot \left(\prod_{i=1}^{(\ell-1)/2} \left(\frac{x \cdot x([i]G) - 1}{x - x([i]G)} \right) \right)^2 . \tag{5}$$

Moving to projective coordinates $(U : V)$ such that $x = U/V$ and using the fact that $\mathcal{X}_G = \{(x([i]G) : 1) : 1 \leq i \leq (\ell - 1)/2\}$, Eq. (5) becomes

$$\phi_x\big((U : V)\big) = (U' : V')$$

where

$$\begin{cases} U' = U \big[\prod_{(X_Q : Z_Q) \in \mathcal{X}_G} (UX_Q - VZ_Q) \big]^2, \\ V' = V \big[\prod_{(X_Q : Z_Q) \in \mathcal{X}_G} (UZ_Q - VX_Q) \big]^2. \end{cases} \tag{6}$$

Algorithm 1 (from [9]) and Algorithm 2 (our space-efficient variant) compute ϕ_x at a series of input points using an efficient evaluation of the expressions in (6). For the moment, we assume that we have subroutines

[5] The algorithms of [9] focus on odd-degree isogenies, treating 2 and 4-isogenies as special cases; Renes [24] extends the approach to even-degree isogenies, and provides a satisfying theoretical framework.

– `KernelPoints` (for Algorithm 1): given $(X_G : Z_G)$, returns $\mathcal{X}_G$ as a list.
– `KernelRange` (for Algorithm 2): a *generator* coroutine which, given $(X_G : Z_G)$, constructs and yields the elements of $\mathcal{X}_G$ to the caller one by one.
– `CrissCross` (for Algorithms 1 and 2, from [9, Algorithm 1]) takes $(\alpha, \beta, \gamma, \delta)$ in $\mathbb{F}_{q^k}^4$ and returns $(\alpha\delta + \beta\gamma, \alpha\delta - \beta\gamma)$ in $\mathbb{F}_{q^k}^2$ at a cost of $2\mathbf{M} + 2\mathbf{a}$.

We discuss algorithms to implement `KernelPoints` and `KernelRange` in §4

Algorithm 1: Combines Algorithms 3 and 4 from [9] to evaluate an ℓ-isogeny of Montgomery models at a list of input points. The total cost is $2n\ell\mathbf{M} + 2n\mathbf{S} + ((n+1)(\ell+1) - 2)\mathbf{a}$, *plus* the cost of `KernelPoints`.

Input: The x-coordinate $(X_G : Z_G)$ of a generator G of the kernel of an ℓ-isogeny ϕ, and a list of evaluation points $((U_i : V_i) : 1 \le i \le n)$
Output: The list of images $((U_i' : V_i') = \phi_x((U_i : V_i)) : 1 \le i \le n)$

$((X_1, Z_1), \ldots, (X_{(l-1)/2}, Z_{(l-1)/2})) \leftarrow$ `KernelPoints`$((X_G : Z_G))$ `// See §4`
for $1 \le i \le (\ell-1)/2$ **do**
  $(\hat{X}_i, \hat{Z}_i) \leftarrow (X_i + Z_i, X_i - Z_i)$ `// 2a`
for $i = 1$ ***to*** n **do**
  $(\hat{U}_i, \hat{V}_i) \leftarrow (U_i + V_i, U_i - V_i)$ `// 2a`
  $(U_i', V_i') \leftarrow (1, 1)$
  for $j = 1$ ***to*** $(\ell-1)/2$ **do**
    $(t_0, t_1) \leftarrow$ `CrissCross`$(\hat{X}_j, \hat{Z}_j, \hat{U}_i, \hat{V}_i)$ `// 2M + 2a`
    $(U_i', V_i') \leftarrow (t_0 \cdot U_i', t_1 \cdot V_i')$ `// 2M`
  $(U_i', V_i') \leftarrow (U_i \cdot (U_i')^2, V_i \cdot (V_i')^2)$ `// 2M + 2S`
return $((U_1', V_1'), \ldots, (U_n', V_n'))$

4 Accelerating Vélu: faster iteration over the kernel

Let $\mathcal{E}/\mathbb{F}_q$ be an elliptic curve, and let G be a point of prime order ℓ in $\mathcal{E}$. For simplicity, in this section we will assume that G is defined over $\mathbb{F}_q$, but all of the results here apply when G is defined over an extension $\mathbb{F}_{q^k}$: in that case, the only change is that $\mathbf{M}$, $\mathbf{S}$, and $\mathbf{a}$ represent operations in the extension field $\mathbb{F}_{q^k}$, while $\mathbf{C}$ represents multiplication of an element of $\mathbb{F}_{q^k}$ by a curve constant of the subfield $\mathbb{F}_q$ (which is roughly k times cheaper than $\mathbf{M}$). We will return to the case where G is defined over an extension in §5, where we can combine results from this section with the action of Frobenius.

4.1 Kernel point enumeration and differential addition chains

We now turn to the problem of enumerating the set $\mathcal{X}_G$. This process, which we call *kernel point enumeration*, could involve constructing the entire set (as in `KernelPoints`) or constructing its elements one by one (for `KernelRange`).

Algorithm 2: A generator-based version of Algorithm 1, with much lower space requirements when $\ell \gg n$. The total cost is $2n\ell\mathbf{M} + 2n\mathbf{S} + (2n + (\ell-1)(n+1))\mathbf{a}$, *plus* the cost of a full run of `KernelRange`.

Input: The x-coordinate $(X_G : Z_G)$ of a generator G of the kernel of an ℓ-isogeny ϕ, and a list of evaluation points $((U_i : V_i) : 1 \le i \le n)$
Output: The list of images $((U_i' : V_i') = \phi_x((U_i : V_i)) : 1 \le i \le n)$

for $1 \le i \le n$ **do**
  $(\hat{U}_i, \hat{V}_i) \leftarrow (U_i + V_i, U_i - V_i)$ `// 2a`
  $(U_i', V_i') \leftarrow (1, 1)$
for $(X : Z)$ ***in*** `KernelRange`$((X_G : Z_G))$ **do** `// See §4`
  $(\hat{X}, \hat{Z}) \leftarrow (X + Z, X - Z)$ `// 2a`
  for $1 \le i \le n$ **do**
    $(t_0, t_1) \leftarrow$ `CrissCross`$(\hat{X}, \hat{Z}, \hat{U}_i, \hat{V}_i)$ `// 2M + 2a`
    $(U_i', V_i') \leftarrow (t_0 \cdot U_i', t_1 \cdot V_i')$ `// 2M`
for $1 \le i \le n$ **do**
  $(U_i', V_i') \leftarrow (U_i \cdot (U_i')^2, V_i \cdot (V_i')^2)$ `// 2M + 2S`
return $((U_1', V_1'), \ldots, (U_n', V_n'))$

For $\ell = 2$ and 3, there is nothing to be done because $\mathcal{X}_G = \{(X_G : Z_G)\}$; so from now on we consider the case $\ell > 3$.

We allow ourselves two curve operations for kernel point enumeration: `xADD` and `xDBL`. In §5, where G is assumed to be defined over a nontrivial extension of the base field, we will also allow the Frobenius endomorphism.

Every algorithm constructing a sequence of elements of $\mathcal{X}_G$ using a series of `xADD` and `xDBL` instructions corresponds to a *modular differential addition chain*.

Definition 2. *A **Modular Differential Addition Chain (MDAC)** for a set $S \subset \mathbb{Z}/\ell\mathbb{Z}$ is a sequence of integers $(c_0, c_1, c_2, \ldots, c_n)$ such that*

1. *every element of S is represented by some $c_i \pmod{\ell}$,*
2. *$c_0 = 0$ and $c_1 = 1$, and*
3. *for each $1 < i \le n$ there exist $0 \le j(i), k(i), d(i) < i$ such that $c_i \equiv c_{j(i)} + c_{k(i)} \pmod{\ell}$ and $c_{j(i)} - c_{k(i)} \equiv c_{d(i)} \pmod{\ell}$.*

Algorithms to enumerate $\mathcal{X}_G$ using `xADD` and `xDBL` correspond to MDACs $(c_0, \ldots, c_n)$ for $\{1, \ldots, (\ell-1)/2\}$: the algorithm starts with $x([c_0]G) = x(0) = (1 : 0)$ and $x([c_1]G) = x(G) = (X_G : Z_G)$, then computes each $x([c_i]G)$ using

$$x([c_i]G) = \begin{cases} \texttt{xADD}(x([c_{j(i)}]G), x([c_{k(i)}]G), x([c_{d(i)}]G)) & \text{if } d(i) \neq 0\,, \\ \texttt{xDBL}([c_{j(i)}]G) & \text{if } d(i) = 0\,. \end{cases}$$

4.2 Additive kernel point enumeration

The classic approach is to compute $\mathcal{X}_G$ using repeated `xADD`s. Algorithm 3 is Costello and Hisil's `KernelPoints` function [9, Algorithm 2]. This corresponds

to the MDAC $(0, 1, 2, 3, \ldots, (\ell-1)/2)$ computed by repeatedly adding 1 (in the notation of Definition 2, $(j(i), k(i), d(i)) = (i-1, 1, i-2)$), except for 2 which is computed by doubling 1. The simplicity of this MDAC means that Algorithm 3 adapts almost trivially to `KernelRange` using a relatively small internal state: in order to generate the next $(X_{i+1} : Z_{i+1})$, we need only keep the values of $(X_i : Z_i)$, $(X_{i-1} : Z_{i-1})$, and $(X_1 : Z_1)$.

Algorithm 3: Basic kernel point enumeration by repeated addition.
Uses exactly 1 `xDBL` and $(\ell-5)/2$ `xADD` operations (for prime $\ell > 3$).

Input: The x-coordinate $(X_G : Z_G)$ of the generator G of a cyclic subgroup of order ℓ in $\mathcal{E}(\mathbb{F}_q)$
Output: $\mathcal{X}_G$ as a list

1 $(X_1 : Z_1) \leftarrow (X_G : Z_G)$
2 $(X_2 : Z_2) \leftarrow \texttt{xDBL}\big((X_G : Z_G)\big)$
3 **for** $i = 3$ ***to*** $(\ell-1)/2$ **do** `// Invariant:` $(X_i : Z_i) = x([i]G)$
4 $\quad (X_i : Z_i) \leftarrow \texttt{xADD}\big((X_{i-1} : Z_{i-1}), (X_G : Z_G), (X_{i-2}, Z_{i-2})\big)$
5 **return** $\big((X_1 : Z_1), \ldots, (X_{(\ell-1)/2} : Z_{(\ell-1)/2})\big)$

4.3 Replacing `xADD`s with `xDBL`s

Comparing x-only operations on Montgomery curves, replacing an `xADD` with an `xDBL` trades $2\mathbf{M}$ and $2\mathbf{a}$ for $1\mathbf{C}$. We would therefore like to replace as many `xADD`s as possible in our kernel enumeration with `xDBL`s.

As a first attempt, we can replace Line 4 of Algorithm 3 with

$$(X_i : Z_i) \leftarrow \begin{cases} \texttt{xDBL}((X_{i/2} : Z_{i/2})) & \text{if } i \text{ is even,} \\ \texttt{xADD}\big((X_{i-1} : Z_{i-1}), (X_G : Z_G), (X_{i-2}, Z_{i-2})\big) & \text{if } i \text{ is odd.} \end{cases}$$

But applying this trick systematically requires storing many more intermediate values, reducing the efficiency of `KernelRange`. It also only replaces half of the `xADD`s with `xDBL`s, and it turns out that we can generally do much better.

4.4 Multiplicative kernel point enumeration

We can do better for a large class of ℓ by considering the quotient

$$M_\ell := \big(\mathbb{Z}/\ell\mathbb{Z}\big)^\times / \langle \pm 1 \rangle \,.$$

(We emphasize that M_ℓ is a quotient of the *multiplicative* group.) For convenience, we write

$$m_\ell := \#M_\ell = (\ell-1)/2 \,.$$

We can now reframe the problem of enumerating $\mathcal{X}_G$ as the problem of enumerating a complete set of representatives for M_ℓ. The MDAC of Algorithm 3

computes the set of representatives $\{1, 2, \ldots, m_\ell\}$, but for the purposes of enumerating $\mathcal{X}_G$, *any* set of representatives will do. Example 1 is particularly useful.

Example 1. Suppose 2 generates M_ℓ. This is the case if 2 is a primitive element modulo ℓ—that is, if 2 has order $(\ell-1)$ modulo ℓ—but also if 2 has order $(\ell-1)/2$ modulo ℓ and $\ell \equiv 3 \pmod 4$. In this case

$$M_\ell = \{2^i \bmod \ell : 0 \le i < m_\ell\},$$

so $(0, 1, 2, 4, 8, \ldots, 2^{m_\ell})$ is an MDAC for M_ℓ that can be computed using *only* doubling, and *no* differential additions.

The `KernelPoints` and `KernelRange` driven by the MDAC of Example 1 replace *all* of the $(\ell-1)/2-2$ `xADD`s in Algorithm 3 with cheaper `xDBL`s: we save $\ell-5$ **M** and $\ell-5$ **a** at the cost of $(\ell-5)/2$ **C**. The `KernelRange` based on this MDAC is particularly simple: each element depends only on its predecessor, so the internal state consists of a single $(X_i : Z_i)$.

So, how often does this trick apply? Theoretically, the quantitative form of Artin's primitive root conjecture (proven by Hooley under GRH) says that $M_\ell = \langle 2 \rangle$ for a little over half of all$'\ell$ (see [31]). Experimentally, 5609420 of the first 10^7 odd primes ℓ satisfy $M_\ell = \langle 2 \rangle$.

One might try to generalize Example 1 to other generators of M_ℓ: for example, if $M_\ell = \langle 3 \rangle$, then we could try to find an MDAC for $\{3^i \bmod \ell : 0 \le i < (\ell-1)/2\}$. But this is counterproductive: x-only tripling (or multiplication by any scalar > 2) is *slower* than differential addition.

4.5 Stepping through cosets

What can we do when $M_\ell \neq \langle 2 \rangle$? A productive generalization is to let

$$A_\ell := \langle 2 \rangle \subseteq M_\ell \quad \text{and} \quad a_\ell := \#A_\ell\,,$$

and to try to compute a convenient decomposition of M_ℓ into cosets of A_ℓ. Within each coset, we can compute elements using repeated `xDBL`s as in Example 1; then, it remains to step from one coset into another using differential additions.

This can be done in a particularly simple way for the primes ℓ such that

$$M_\ell \text{ is generated by 2 and 3.} \qquad (*)$$

If $(*)$ holds, then

$$M_\ell = \bigsqcup_{i=0}^{m_\ell/a_\ell - 1} 3^i A_\ell\,.$$

We can move from the i-th to the $(i+1)$-th coset using the elementary relations

$$\begin{cases} c \cdot 2^{j+1} + c \cdot 2^j = 3c \cdot 2^j \\ c \cdot 2^{j+1} - c \cdot 2^j = c \cdot 2^j \end{cases} \qquad \text{for all integers } c \text{ and } j \ge 0. \qquad (7)$$

In particular, if we have enumerated some coset $3^i A_\ell$ by repeated doubling, then we can compute an element of $3^{i+1}A_\ell$ by applying a differential addition to any two consecutive elements of $3^i A_\ell$ (and the difference is the first of them). Algorithm 4 minimises storage overhead by using the last two elements of the previous coset to generate the first element of the next one. The `KernelRange` of Algorithm 4 therefore has an internal state of only two x-coordinates—so not only is it faster than the `KernelRange` of Algorithm 3 where it applies, but it also has a smaller memory footprint.

Algorithm 4: Kernel enumeration for $\ell > 3$ satisfying $(*)$. Cost: $(1 - 1/a_\ell) \cdot m_\ell$ `xDBL`s and $m_\ell/a_\ell - 1$ `xADD`s.

Input: Projective x-coordinate $(X_G : Z_G)$ of the generator G of a cyclic subgroup of order ℓ in $\mathcal{E}(\mathbb{F}_q)$, where ℓ satisfies $(*)$.
Output: $\mathcal{X}_G$ as a list

$(a, b) \leftarrow (a_\ell, m_\ell/a_\ell)$
for $i = 0$ ***to*** $b - 1$ **do** `// Invariant:` $(X_{ai+j} : Z_{ai+j}) = x([3^i 2^{i(a-2)+(j-1)}]G)$
  if $i = 0$ **then**
    $(X_1 : Z_1) \leftarrow (X_G : Z_G)$
  else `// Compute new coset representative`
    $(X_{ai+1} : Z_{ai+1}) \leftarrow \texttt{xADD}\big((X_{ai} : Z_{ai}), (X_{ai-1} : Z_{ai-1}), (X_{ai-1} : Z_{ai-1})\big)$
  for $j = 2$ ***to*** a **do** `// Exhaust coset by doubling`
    $(X_{ai+j} : Z_{ai+j}) \leftarrow \texttt{xDBL}\big((X_{ai+j-1} : Z_{ai+j-1})\big)$
return $\big((X_1 : Z_1), \ldots, (X_{(\ell-1)/2} : Z_{(\ell-1)/2})\big)$

Algorithm 4 performs better the closer a_ℓ is to m_ℓ. In particular, when $A_\ell = M_\ell$, it uses $m_\ell - 1$ `xDBL`s and no `xADD`s at all. The worst case for Algorithm 4 is when the order of 2 in M_ℓ is as small as possible: that is, when $\ell = 2^k - 1$. In this case $a_\ell = k$, and compared with Algorithm 3 we still reduce the number of `xADD`s to be done by a factor of k.

4.6 The remaining primes

While 1878 of the 2261 odd primes $\ell \leq 20000$ satisfy $(*)$, there are still 383 primes that do not. We can, to some extent, adapt Algorithm 4 to handle these primes, but on a case-by-case basis and with somewhat less satisfactory results.

For example, the CSIDH-512 parameter set specifies 74 isogeny-degree primes

$$\ell = 3, 5, 7, 11, 13, \ldots, 367, 373, \text{ and } 587\,.$$

Of these 74 primes, all but seven satisfy $(*)$: the exceptions are $\ell = 73$, 97, 193, 241, 313, and 337. Table 1 lists these primes and a candidate decomposition of M_ℓ into cosets of A_ℓ. In each case, we need to produce either an element of $5A_\ell$ or $7A_\ell$. This can certainly be done using previously-computed elements, but it

requires careful tracking of those elements, which implies a larger internal state and a more complicated execution pattern, ultimately depending on the value of ℓ.

Table 1. Primes ℓ in the CSIDH-512 parameter set that do not satisfy $(*)$.

Prime ℓ	a_ℓ	$[M_\ell : \langle 2,3 \rangle]$	Coset decomposition of M_ℓ	Notes
73	9	2	$M_{73} = A_{73} \sqcup 3A_{73} \sqcup 5A_{73} \sqcup 5 \cdot 3A_{73}$	
97	24	2	$M_{97} = A_{97} \sqcup 5A_{97}$	3 is in A_{97}
193	48	2	$M_{193} = A_{193} \sqcup 5A_{193}$	3 is in A_{193}
241	12	2	$M_{241} = \left(\bigsqcup_{i=0}^{4} 3^i A_{241}\right) \sqcup \left(\bigsqcup_{i=0}^{4} 7 \cdot 3^i A_{241}\right)$	
307	51	3	$M_{307} = A_{307} \sqcup 5A_{307} \sqcup 7A_{307}$	3 is in A_{313}
313	78	2	$M_{313} = A_{313} \sqcup 5A_{313}$	3 is in A_{193}
337	21	2	$M_{337} = \left(\bigsqcup_{i=0}^{3} 3^i A_{337}\right) \sqcup \left(\bigsqcup_{i=0}^{3} 5 \cdot 3^i A_{337}\right)$	

Example 2. Consider $\ell = 97$. In this case, 3 is in A_{97} (in fact $3 \equiv 2^{19} \pmod{97}$), and we find that $M_{97} = A_{97} \sqcup 5A_{97}$.

To adapt Algorithm 4 to this case, we can still enumerate A_{97} using repeated doubling. Then, we need to construct an element of $5A_{97}$ from elements of A_{97}, for which we can use a differential addition like $5 \cdot 2^i = 2^{i+2} + 2^i$ (with difference $3 \cdot 2^i$) or $5 \cdot 2^i = 2^{i+1} + 3 \cdot 2^i$ (with difference 2^i). Each involves near powers of 2 (modulo 97), but also $3 \cdot 2^i$—which we know is in A_{97}, so it need not be recomputed, but we need to know that $3 \cdot 2^i \equiv 2^{i+19} \pmod{97}$ so that we can identify and store the x-coordinate corresponding to 3^i (for a chosen i) while enumerating A_{97}. The end result is an algorithm that uses one `xADD` and 48 `xDBL`s, just like Algorithm 4, but the internal state is slightly larger and the more complicated execution pattern specific to $\ell = 97$.

Alternatively, after (or while) enumerating A_{97}, we could just recompute $3 = 1 + 2$ (difference 1) to get 5 as $1 + 4$ (difference 3) or $2 + 3$ (difference 1), but this recomputation of 3 is redundant.

Ultimately, there does not seem to be any "one size fits all" generalization of Algorithm 4 for enumerating $\mathcal{X}_G$ without either a more complicated state or redundant recomputations. The obvious approach of finding an MDAC to enumerate a set of representatives for $M_\ell / \langle 2,3 \rangle$ and then using Algorithm 4 to exhaust the coset containing each representative can give reasonable results for many ℓ not satisfying $(*)$, but the savings are generally not optimal.

4.7 (In)Compatibility with Vélusqrt

One natural question is whether these techniques can be used to further accelerate the Vélusqrt algorithm of [2], which can evaluate isogenies of large prime

degree ℓ in $\widetilde{O}(\sqrt{\ell})$ (with $O(\sqrt{\ell})$ space). Vélusqrt never explicitly computes all of $\mathcal{X}_G$. Instead, it relies on the existence of a decomposition

$$S := \{1, 3, 5, \ldots, \ell - 2\} = (I + J) \sqcup (I - J) \sqcup K \tag{8}$$

where I, J, and K are sets of integers of size $O(\sqrt{\ell})$ such that the maps $(i, j) \to i + j$ and $(i, j) \to i - j$ are injective with disjoint images. In [2], these sets are

$$\begin{aligned} I &:= \{2b(2i + 1) : 0 \leq i < b'\}, \\ J &:= \{2j + 10 \leq j < b\}, \\ K &:= \{4bb' + 1, \ldots, \ell - 4, \ell - 2\} \end{aligned}$$

where $b := \lfloor \sqrt{\ell - 1}/2 \rfloor$ and $b' := \lfloor (\ell - 1)/4b \rfloor$. (Note that I contains "giant steps", J contains "baby steps", and K contains the rest of S).

The key thing to note here is that this decomposition is essentially additive, and the elements of I, J, and K form arithmetic progressions. Algorithm 4, however, is essentially multiplicative: it works with subsets in geometric progression, not arithmetic progression. We cannot exclude the existence of subsets I, J, and K of size $O(\sqrt{\ell})$ satisfying Equation (8) and which are amenable to enumeration by 2-powering or a variation of Algorithm 4 for some, or even many ℓ, but it seems difficult to construct nontrivial and useful examples.

5 Irrational kernel points: exploiting Frobenius

Now suppose G is defined over a nontrivial extension $\mathbb{F}_{q^k}$ of $\mathbb{F}_q$, but $\langle G \rangle$ is defined over the subfield $\mathbb{F}_q$: that is, it is Galois-stable. In particular, the q-power Frobenius endomorphism π of $\mathcal{E}$, which maps points in $\mathcal{E}(\mathbb{F}_{q^k})$ to their conjugates under $\mathrm{Gal}(\mathbb{F}_{q^k}/\mathbb{F}_q)$, maps $\langle G \rangle$ into $\langle G \rangle$. In Appendix A we show how we can find the point G.

Since π maps $\langle G \rangle$ into $\langle G \rangle$, it restricts to an endomorphism of $\langle G \rangle$—and since the endomorphisms of $\langle G \rangle$ are $\mathbb{Z}/\ell\mathbb{Z}$, and Frobenius has no kernel (so π is not 0 on $\langle G \rangle$), it must act as multiplication by an eigenvalue $\lambda \neq 0$ on $\langle G \rangle$. The precise value of λ is not important here, but we will use the fact that λ has order k in $(\mathbb{Z}/\ell\mathbb{Z})^\times$ and order k' in $(\mathbb{Z}/\ell\mathbb{Z})^\times/\langle \pm 1 \rangle$.

Now let

$$F_\ell := \langle \lambda \rangle \subseteq M_\ell \qquad \text{and} \qquad c_F := [M_\ell : F_\ell] = \frac{m_\ell}{k'}.$$

Let R_0 be a set of representatives for the cosets of F_ℓ in M_ℓ, and let $S_0 = \{[r]G : r \in R_0\}$. Note that

$$\#S_0 = (\ell - 1)/k'.$$

5.1 The cost of Frobenius

In this section, we seek to use the Galois action to replace (many) **M** and **S** with a few **F**. For this to be worthwhile, **F** must be cheap: and they are, even

if this is not obvious given the definition of the Frobenius map on $\mathbb{F}_{q^k}$ as q-th powering. It is important to note that we do not compute Frobenius by powering. Instead, we use the fact that Frobenius is an $\mathbb{F}_q$-linear map on $\mathbb{F}_{q^k}$ viewed as an $\mathbb{F}_q$-vector space: that is, Frobenius acts as a $k \times k$ matrix (with entries in $\mathbb{F}_q$) on the coefficient vectors of elements in $\mathbb{F}_{q^k}$.

The form of the Frobenius matrix, and the cost of applying it, depends on the basis of $\mathbb{F}_{q^k}/\mathbb{F}_q$. For example:

1. If $k = 2$ and $\mathbb{F}_{q^2} = \mathbb{F}_q(\sqrt{\Delta})$, then Frobenius simply negates $\sqrt{\Delta}$ and the matrix is $\mathrm{diag}(1, -1)$, so $\mathbf{F} \approx 0$.
2. If $\mathbb{F}_{q^k}/\mathbb{F}_q$ is represented with a normal basis, then the matrix represents a cyclic permutation, and again $\mathbf{F} \approx 0$.

Even in the worst case where the basis of $\mathbb{F}_{q^k}/\mathbb{F}_q$ has no special Galois structure, $\mathbf{F}$ is just the cost of multiplying a k-vector by a $k \times k$ matrix over $\mathbb{F}_q$: that is, k^2 $\mathbb{F}_q$-multiplications and $k(k-1)$ $\mathbb{F}_q$-additions. This is close to the cost of a single $\mathbb{F}_{q^k}$-muliplication using the "schoolbook" method; so when $k \leq 12$, we have $\mathbf{F} \approx \mathbf{M}$ in the worst case.

5.2 Galois orbits

Each point $P \in \mathcal{E}(\mathbb{F}_{q^k})$ is contained in a *Galois orbit* containing all the conjugates of P. The kernel subgroup $\langle G \rangle$ breaks up (as a set) into *Galois orbits*: if we write

$$\mathcal{O}_P := \{P, \pi(P), \ldots, \pi^{k-1}(P)\} \qquad \text{for } P \in \mathcal{E}(\mathbb{F}_{q^k}),$$

then

$$\langle G \rangle = \{0\} \sqcup \begin{cases} \bigsqcup_{P \in S_0} \mathcal{O}_P & \text{if } k \text{ is even,} \\ \left(\bigsqcup_{P \in S_0} \mathcal{O}_P\right) \sqcup \left(\bigsqcup_{P \in S_0} \mathcal{O}_{-P}\right) & \text{if } k \text{ is odd.} \end{cases} \tag{9}$$

To get a picture of where we are going, recall from §3 that in general, isogeny evaluation can be reduced to evaluations of the kernel polynomial

$$D(X) := \prod_{P \in S} (X - x(P)),$$

where $S \subset \langle G \rangle$ is any subset such that $S \cap -S = \emptyset$ and $S \cup -S = \langle G \rangle \setminus \{0\}$. The decomposition of (9) can be seen in the factorization of $D(X)$ over $\mathbb{F}_{q^k}$:

$$\begin{aligned} D(X) = \prod_{P \in S} (X - x(P)) &= \prod_{P \in S_0} \prod_{i=0}^{k'-1} (X - x(\pi^i(P))) \\ &= \prod_{P \in S_0} \prod_{i=0}^{k'-1} (X - x(P)^{q^i}), \end{aligned}$$

and the factors corresponding to each P in S_0 are the irreducible factors of D over $\mathbb{F}_q$. Transposing the order of the products, if we let

$$D_0(X) := \prod_{P \in S_0} (X - x(P))$$

then for α in the base field $\mathbb{F}_q$, we can compute $D(\alpha)$ by computing $D_0(\alpha)$ and taking the norm:

$$D(\alpha) = \mathrm{Norm}(D_0(\alpha)) \quad \text{for all } \alpha \in \mathbb{F}_q\,.$$

where

$$\mathrm{Norm}(x) := \prod_{i=0}^{k'-1} x^{q^i} = x(x(\cdots(x(x)^q)^q\cdots)^q)^q\,,$$

which can be computed for the cost of $(k-1)\mathbf{F}+(k-1)\mathbf{M}$ (some multiplications can be saved with more storage, but for small k this may not be worthwhile).

Similarly, we can rewrite the rational map ϕ_x from (5) as

$$\phi_x(x) = x\cdot\left(\prod_{P\in S}\left(\frac{x\cdot x(P)-1}{x-x(P)}\right)\right)^2 = x\cdot\left(\prod_{P\in S_0}\prod_{i=0}^{k'-1}\left(\frac{x\cdot x(P)^{q^i}-1}{x-x(P)^{q^i}}\right)\right)^2.$$

Evaluating ϕ_x at α in $\mathbb{F}_q$, rearranging the products gives

$$\phi_x(\alpha) = \alpha\cdot\left(\prod_{P\in S_0}\prod_{i=0}^{k'-1}\left(\frac{\alpha\cdot x(P)^{q^i}-1}{\alpha-x(P)^{q^i}}\right)\right)^2 = \alpha\cdot\mathrm{Norm}(\overline{\phi}_x(\alpha))^2\,,$$

where

$$\overline{\phi}_x(X) := \prod_{P\in S_0}\frac{X\cdot x(\pi^i(P))-1}{X-x(\pi^i(P))}$$

Projectively, from (6) we get $\phi_x : (U:V)\mapsto(U':V')$ where

$$U' = U\cdot\Big[\prod_{i=0}^{k'-1}\prod_{P\in S_0}(UX_P^{q^i}-Z_P^{q^i}V)\Big]^2,$$

$$V' = V\cdot\Big[\prod_{i=0}^{k'-1}\prod_{P\in S_0}(UZ_P^{q^i}-X_P^{q^i}V)\Big]^2,$$

so if we set

$$F(U,V) := \prod_{P\in S_0}(U\cdot X_P - Z_P\cdot V) \quad\text{and}\quad G(U,V) := \prod_{P\in S_0}(U\cdot Z_P - X_P\cdot V)\,,$$

then for α and β in $\mathbb{F}_q$ we get

$$\phi_x((\alpha:\beta)) = (\alpha':\beta') := \Big(\alpha\cdot\mathrm{Norm}(F(\alpha,\beta))^2 : \beta\cdot\mathrm{Norm}(G(\alpha,\beta))^2\Big)\,.$$

5.3 Enumerating representatives for the Galois orbits.

We now need to enumerate a set S_0 of representatives for the Galois orbits modulo ±1 or, equivalently, a set of representatives R_0 for the cosets of F_ℓ in M_ℓ. We therefore want good MDACs for M_ℓ/F_ℓ.

Given an MDAC driving enumeration of the coset representatives, there are obvious adaptations of Algorithms 1 and 2 to this extension field case. Rather than iterating over all of the kernel x-coordinates, we just iterate over a subset representing the cosets of $\mathbb{F}_\ell$, and then compose with the norm.

Concretely, in Algorithm 2, we should

1. Replace `KernelRange` in Line 4 with a generator driven by an efficient MDAC for M_ℓ/F_ℓ;
2. Replace Line 10 with $(U_i', V_i') \leftarrow (U_i \cdot \mathrm{Norm}(U_i')^2, V_i \cdot \mathrm{Norm}(V_i')^2)$.

First, we can consider Algorithm 3: that is, enumerating M_ℓ/F_ℓ by repeated addition. Unfortunately, we do not have a nice bound on the length of this MDAC: the coset representatives are not necessarily conveniently distributed over M_ℓ, so we could end up computing a lot of redundant points.

Example 3. Take $(\ell, k) = (89, 11)$. We see that $M_\ell \neq \langle \lambda, 2 \rangle$. So we compute the minimal element in each Galois orbit (up to negation), and choose it to be our orbit representative. Using arithmetic modulo 89 only, we found a choice for R_0 to be $R_0 = \{1, 3, 5, 13\}$. Now we compute an optimal MDAC, namely $(0, 1, 2, 3, 5, 8, 13)$. This chain computes the orbit generator x-coordinates using one `xDBL` operation and six `xADD` operations, albeit involving the computation of two intermediate points that will not be utilized in the final result.

But when we say that the coset representatives are not conveniently distributed over M_ℓ, we mean that with respect to addition. If we look at M_ℓ multiplicatively, then the path to efficient MDACs is clearer.

The nicest case is when $M_\ell = \langle 2, \lambda \rangle$: then, we can take $R_0 = \{2^i : 0 \leq i < c_F\}$, which brings us to the 2-powering MDAC of Example 1—except that we stop after $c_F - 1$ `xDBL`s. We thus reduce the number of `xDBL`s by a factor of $\approx k'$, at the expense of two norm computations.

This MDAC actually applies to more primes ℓ here than it did in §4, because we no longer need 2 to generate all of M_ℓ; we have λ to help. In fact, the suitability of this MDAC no longer depends on ℓ, but also on k.

We can go further if we assume

$$M_\ell = \langle 2, 3, \lambda \rangle \,. \tag{$**$}$$

To simplify notation, we define

$$a_{\ell,k} := [\langle 2, \lambda \rangle : F_\ell] \,, \qquad b_{\ell,k} := [\langle 2, 3, \lambda \rangle : \langle 2, \lambda \rangle] = c_F / a_{\ell,k} \,.$$

Algorithm 5 is a truncated version of Algorithm 4 for computing S_0 instead of $\mathcal{X}_G$ when $(**)$ holds. Algorithm 6 is the corresponding modification of Algorithm 1, evaluating an ℓ-isogeny over $\mathbb{F}_q$ with kernel $\langle G \rangle$ at n points of $\mathcal{E}(\mathbb{F}_q)$, where $x(G)$ is in $\mathbb{F}_{q^{k'}}$ with $k' > 1$.

Table 2 compares the total cost of Algorithms 6 and 5 with that of Algorithms 1 and 3. In both algorithms, we can take advantage of the fact that many of the multiplications have one operand in the smaller field $\mathbb{F}_q$: notably, the

multiplications involving coordinates of the evaluation points. In the context of isogeny-based cryptography (where curve constants look like random elements of $\mathbb{F}_q$), this means that in Algorithm 1, we can replace the $2\mathbf{M} + 2\mathbf{a}$ in Line 8 and the $2\mathbf{M} + 2\mathbf{S}$ in Line 10 with $2\mathbf{C} + 2\mathbf{a}$ and $2\mathbf{C} + 2\mathbf{S}$, respectively.

Table 2. Comparison of ℓ-isogeny evaluation algorithms for kernels $\langle G \rangle$ defined over $\mathbb{F}_q$ but with $x(G) \in \mathbb{F}_{q^{k'}}$. In this table, $\mathbf{C}$ denotes multiplications of elements of $\mathbb{F}_{q^{k'}}$ by elements of $\mathbb{F}_q$ (including, but not limited to, curve constants).

	Costello–Hisil (Algorithms 1 and 3)	This work (Algorithm 6)
M	$(\ell-1)n + 2\ell - 8$	$2(c_F + k' - 1)n + 2c_F + 2b_{\ell,k} + 4$
S	$2n + \ell - 3$	$2n + 2c_F - 2$
C	$(\ell+1)n + 1$	$2(c_F + 1)n + c_F - b_{\ell,k}$
a	$(n+1)(\ell+1) + 3\ell + 17$	$2c_F(n+1) + 4c_F + 4b_{\ell,k} - 6$
F	0	$2(k'-1)n$

Algorithm 5: Compute S_0 when $(**)$ holds. Cost: $b_{\ell,k} - 1$ `xADD`s and $(c_F - b_{\ell,k})$ `xDBL`s, or $(2c_F + 2b_{\ell,k} + 4)\mathbf{M} + (2c_F - 2)\mathbf{S} + (c_F - b_{\ell,k})\mathbf{C} + (4c_F + 4b_{\ell,k} - 6)\mathbf{a}$

Input: Projective x-coordinate $(X_G : Z_G)$ of the generator G of a cyclic subgroup of order ℓ in $\mathcal{E}(\mathbb{F}_{q^k})$, where ℓ satisfies $M_\ell = \langle 2, 3, \lambda \rangle$.
Output: S_0 as a list

```
1  Function SZeroPoints((X_G : Z_G))
2      (a, b) ← (a_{ℓ,k}, b_{ℓ,k})
3      for i = 0 to b − 1 do          // Invariant: x_{ai+j} = x([3^i 2^{i(a−2)+(j−1)}]G)
4          if i = 0 then
5              x_1 ← (X_G : Z_G)
6          else                                  // Compute new coset representative
7              x_{ai+1} ← xADD(x_{ai}, x_{ai−1}, x_{ai−1})
8          for j = 2 to a do                     // Exhaust coset by doubling
9              x_{ai+j} ← xDBL(x_{ai+j−1})
10     return (x_1, ..., x_{c_F})
```

If the parameter choice for (ℓ, k) does not satisfy any of these criteria, then we have to compute the coset representatives using some ad-hoc MDAC. We can do some precomputations here to determine an optimal, or near optimal, approach to computing R_0.

Algorithm 6: Isogeny evaluation using `SZeroPoints` and Frobenius. Cost: $2(c_F+k-1)n\mathbf{M}+2n\mathbf{S}+2(cF+1)n\mathbf{C}+2c_F(n+1)\mathbf{a}+2(k-1)n\mathbf{F}$ *plus* the cost of `SZeroPoints`.

Input: The x-coordinate $(X_G : Z_G)$ of a generator G of the kernel of an ℓ-isogeny ϕ, and a list of evaluation points $((U_i : V_i) : 1 \le i \le n)$
Output: The list of images $((U_i' : V_i') = \phi_x((U_i : V_i)) : 1 \le i \le n)$

$((X_1, Z_1), \ldots, (X_{c_F}, Z_{c_F})) \leftarrow$ `SZeroPoints`$((X_G : Z_G))$ `// Algorithm 5`
for $1 \le i \le c_F$ **do**
  $(\hat{X}_i, \hat{Z}_i) \leftarrow (X_i + Z_i, X_i - Z_i)$ `// 2a`
for $i = 1$ ***to*** n **do**
  $(\hat{U}_i, \hat{V}_i) \leftarrow (U_i + V_i, U_i - V_i)$ `// 2a`
  $(U_i', V_i') \leftarrow (1, 1)$
  for $j = 1$ ***to*** c_F **do**
    $(t_0, t_1) \leftarrow$ `CrissCross`$(\hat{X}_j,\ \hat{Z}_j,\ \hat{U}_i,\ \hat{V}_i)$ `// 2C + 2a`
    $(U_i', V_i') \leftarrow (t_0 \cdot U_i', t_1 \cdot V_i')$ `// 2M`
  $(U_i', V_i') \leftarrow$ (`Norm`(U_i'), `Norm`$(V_i'))$ `// `$2(k'-1)\mathtt{M} + 2(k'-1)\mathtt{F}$
  $(U_i', V_i') \leftarrow (U_i \cdot (U_i')^2, V_i \cdot (V_i')^2)$ `// 2C + 2S`
return $((U_1', V_1'), \ldots, (U_n', V_n'))$

5.4 Experimental results

We provide proof-of-concept implementations of our algorithms in SageMath.[6] Our implmentations include operation-counting code to verify the counts claimed in this article. We provide the number of operations for a given ℓ-isogeny and the extension field k. Table 3 displays the costs for our algorithm, highlighted in light gray, compared with the basic Costello–Hisil algorithm (Algorithms 1 and 3). As depicted in Table 3, our approach consistently employs fewer operations across all values of ℓ and extension fields. For k of this size, it is reasonable to use the approximation $\mathbf{F} \approx \mathbf{M}$ (see §5.1).

Finally, Table 4 shows our success rate (over all primes $\ell < 10^4$) at finding optimal MDACs for $k \le 12$. These rates are computed by choosing the minimal representative of each Galois orbit to be in S_0, and checking whether the set S_0 can be computed without any intermediary additions (that would not be otherwise used). Note, this computation checks only one approach for computing the MDAC, hence the percentages in Table 4 represent a lower bound on the number of ℓ that have an optimal MDAC.

6 Applications

Our algorithms have potential applications in any isogeny-based cryptosystem involving isogenies of prime degree $\ell > 3$, including key exchanges like CSIDH [5]

[6] Sage scripts available from `https://github.com/vgilchri/k-velu`.

Table 3. Cost of evaluating an ℓ-isogeny at a single point over $\mathbb{F}_q$, using a kernel generator with x-coordinate in $\mathbb{F}_{q^{k'}}$.

ℓ	k'	**M**	**S**	**C**	**a**	**F**	Algorithm
13	any	30	12	15	54	0	Costello–Hisil (Algorithm 1 with 3)
	1	22	12	19	46	0	This work (Algorithm 6)
	3	10	4	7	14	4	This work (Algorithm 6)
19	any	48	18	21	84	0	Costello–Hisil (Algorithm 1 with 3)
	1	34	18	28	70	0	This work (Algorithm 6)
	3	14	6	10	22	4	This work (Algorithm 6)
	9	18	2	4	6	16	This work (Algorithm 6)
23	any	60	22	25	104	0	Costello–Hisil (Algorithm 1 with 3)
	1	42	22	34	86	0	This work (Algorithm 6)
	11	22	2	4	6	20	This work (Algorithm 6)

Table 4. Percentage of primes, $3 \leq \ell < 10^4$ for which an optimal MDAC definitely exists (using the naive choice of S_0).

k	1	2	3	4	5	6	7	8	9	10	11	12
%	100	100	100	100	84	86	76	67	60	56	45	42

and signature schemes such as SQISign [18, 19], SeaSign [15], and CSI-FiSh [4]. We focus on key exchange here, but similar discussion applies for other schemes.

As mentioned in §1, we also have cryptanalytic applications: state-of-the-art attacks on group-action cryptosystems like CSIDH involve computing a massive number of ℓ-isogenies (in order to do a Pollard-style random walk, for example, or a baby-step giant-step algorithm as in [8]). In this context, even minor savings in individual ℓ-isogenies quickly add up to substantial overall savings.

6.1 CSIDH and constant-time considerations

CSIDH is a post-quantum non-interactive key exchange based on the action of the class group of the imaginary quadratic order $\mathbb{Z}[\sqrt{-p}]$ on the set of supersingular elliptic curves $\mathcal{E}/\mathbb{F}_p$ with $\mathrm{End}_{\mathbb{F}_p}(\mathcal{E}) \cong \mathbb{Z}[\sqrt{-p}]$. The action is computed via compositions of ℓ_i-isogenies for a range of small primes $(\ell_1, \ldots, \ell_m)$.

CSIDH works over prime fields $\mathbb{F}_p$, so the methods of §5 do not apply; but Algorithm 4 may speed up implementations at least for the ℓ_i satisfying $(*)$. (We saw in §4.6 that 67 of the 74 primes ℓ_i in the CSIDH-512 parameter set met $(*)$).

The extent of any speedup depends on two factors. The first is the number of evaluation points. Costello and Hisil evaluate at a 2-torsion point other than $(0, 0)$ in order to interpolate the image curve. The constant-time CSIDH of [21] evaluates at one more point (from which subsequent kernels are derived)—that is, $n = 2$; We find $n = 3$ in [23], and [7] discusses $n > 3$. For larger n, the cost

of Algorithm 1 overwhelms kernel enumeration, but our results may still make a simple and interesting improvement when n is relatively small.

The second factor is the organisation of primes into batches for constant-time CSIDH implementations. CTIDH [1] makes critical use of the so-called *Matryoshka* property of isogeny computations to hide the degree ℓ: using Algorithms 1 and 3, ℓ_i-isogeny evaluation is a sub-computation of ℓ_j-isogeny computation whenever $\ell_i < \ell_j$. Organising primes into similar-sized batches, we can add dummy operations to disguise smaller-degree isogenies as isogenies of the largest degree in their batch.

Our Algorithm 4 has a limited Matryoshka property: ℓ_i-isogenies are sub-computations of ℓ_j-isogenies if $a_{\ell_i} \leq a_{\ell_k}$ and $m_{\ell_i}/a_{\ell_i} \leq m_{\ell_j}/a_{\ell,j}$. For constant-time implementations, it would make more sense to make all primes in a batch satisfying $(*)$ a sub-computation of an algorithm using the maximum a_ℓ and maximum m_ℓ/a_ℓ over ℓ in the batch. Redistributing batches is a delicate matter with an important impact on efficiency; therefore, while our work improves the running time for a fixed ℓ, its impact on batched computations remains uncertain, and ultimately depends on specific parameter choices.

6.2 CRS key exchange

The historical predecessors of CSIDH, due to Couveignes [11] and Rostovtsev and Stolbunov [25, 28, 29] are collectively known as CRS. Here the group is the class group of a quadratic imaginary order, acting on an isogeny (sub)class of elliptic curves with that order as their endomorphism ring; the action is computed using a composition of ℓ_i-isogenies for a range of small primes $(\ell_1, \ldots, \ell_m)$.

In [11, 25, 28, 29], isogenies are computed by finding roots of modular polynomials; this makes key exchange extremely slow at reasonable security levels. Performance was greatly improved in [16] using Vélu-style isogeny evaluation, but this requires finding ordinary isogeny classes over $\mathbb{F}_p$ with rational ℓ_i-torsion points over $\mathbb{F}_{q^{k_i}}$ with k_i as small as possible for as many ℓ_i as possible.

One such isogeny class over a 512-bit prime field is proposed in [16, §4]: the starting curve is $\mathcal{E}/\mathbb{F}_p : y^2 = x(x^2 + Ax + 1)$ where $p := 7\prod_\ell \ell - 1$ where the product is over all primes $2 \leq \ell < 380$; and

$$A = \begin{smallmatrix}10861338504649280383859950140772947007703646408372831934324660566888732797778 9\\ 32142488253565145603672591944602210571423767689240032829444439469242521864171\end{smallmatrix}.$$

This curve has rational ℓ-isogenies with rational kernel generators for $\ell = 3$, 5, 7, 11, 13, 17, 103, 523, and 821, and irrational generators over $\mathbb{F}_{q^k}$ for $\ell = 19$, 29, 31, 37, 61, 71, 547, 661, 881, 1013, 1181, 1321, and 1693; these "irrational" ℓ are an interesting basis of comparison for our algorithms: all but 1321 satisfy $(**)$.

Table 5 compares operation counts for Algorithms 1 and 3 against Algorithm 6, which encapsulates the improvements in §5, for ℓ-isogeny evaluation with kernel generators over $\mathbb{F}_{q^k}$ (and arbitrary n), for all of the "irrational" ℓ above *except* 1321. We see that there are substantial savings to be had for all n.

Table 5. Comparison of Costello–Hisil (Algorithms 1 and 3, in white) with our approach (Algorithm 6, in grey) for the CRS parameters with $k > 1$ proposed in [16]. The prime $\ell = 1321$ with $k = 5$ is omitted, since in this case $M_\ell \neq \langle 2, 3, \lambda \rangle$. In each row, **M**, **S**, **a**, and **F** refer to operations on elements of $\mathbb{F}_{q^{k'}}$, while **C** denotes multiplications of elements of $\mathbb{F}_{q^{k'}}$ by elements of $\mathbb{F}_q$ (including, but not limited to, curve constants).

Parameters				Operations				
k	ℓ	$a_{\ell,k}$	$b_{\ell,k}$	**M**	**S**	**C**	**a**	**F**
3	19	3	1	18n + 30	2n + 16	20n + 1	20n + 64	0
				10n + 4	2n + 4	8n + 2	8n + 14	4n
	661	110	1	660n + 1314	2n + 658	662n + 1	662n + 2632	0
				224n + 218	2n + 218	222n + 109	222n + 656	4n
4	1013	23	11	1012n + 2018	2n + 1010	1014n + 1	1014n + 4040	0
				48n + 524	2n + 504	48n + 242	48n + 1074	2n
	1181	59	5	1180n + 2354	2n + 1178	1182n + 1	1182n + 4712	0
				120n + 596	2n + 588	120n + 290	120n + 1302	2n
5	31	1	3	30n + 54	2n + 28	32n + 1	32n + 112	0
				10n + 8	2n + 4	4n	4n + 14	8n
	61	6	1	60n + 114	2n + 58	62n + 1	62n + 232	0
				20n + 10	2n + 10	14n + 5	14n + 32	8n
7	29	2	1	28n + 50	2n + 26	30n + 1	30n + 104	0
				16n + 2	2n + 2	6n + 1	6n + 8	12n
	71	5	1	70n + 134	2n + 68	72n + 1	72n + 272	0
				22n + 8	2n + 8	12n + 4	12n + 26	12n
	547	39	1	546n + 1086	2n + 544	548n + 1	548n + 2176	0
				90n + 76	2n + 76	80n + 38	80n + 230	12n
8	881	55	2	880n + 1754	2n + 878	882n + 1	882n + 3512	0
				116n + 220	2n + 218	112n + 108	112n + 548	6n
9	37	2	1	36n + 66	2n + 34	38n + 1	38n + 136	0
				20n + 2	2n + 2	6n + 1	6n + 8	16n
	1693	94	1	1692n + 3378	2n + 1690	1694n + 1	1694n + 6760	0
				204n + 186	2n + 186	190n + 93	190n + 560	16n

References

1. Gustavo Banegas, Daniel J. Bernstein, Fabio Campos, Tung Chou, Tanja Lange, Michael Meyer, Benjamin Smith, and Jana Sotáková. CTIDH: faster constant-time CSIDH. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2021(4):351–387, 2021.
2. Daniel J. Bernstein, Luca De Feo, Antonin Leroux, and Benjamin Smith. Faster computation of isogenies of large prime degree. In Steven D. Galbraith, editor, *Proceedings of the Fourteenth Algorithmic Number Theory Symposium*, pages 39–55. Mathematics Sciences Publishers, 2020. `https://eprint.iacr.org/2020/341`.
3. Daniel J. Bernstein, Tanja Lange, Chloe Martindale, and Lorenz Panny. Quantum circuits for the CSIDH: optimizing quantum evaluation of isogenies. In Yuval Ishai and Vincent Rijmen, editors, *Advances in Cryptology - EUROCRYPT 2019 - 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, May 19-23, 2019, Proceedings, Part II*, volume 11477 of *Lecture Notes in Computer Science*, pages 409–441. Springer, 2019.
4. Ward Beullens, Thorsten Kleinjung, and Frederik Vercauteren. Csi-fish: Efficient isogeny based signatures through class group computations. In Steven D. Galbraith and Shiho Moriai, editors, *Advances in Cryptology - ASIACRYPT 2019 - 25th International Conference on the Theory and Application of Cryptology and Information Security, Kobe, Japan, December 8-12, 2019, Proceedings, Part I*, volume 11921 of *Lecture Notes in Computer Science*, pages 227–247. Springer, 2019.
5. Wouter Castryck, Tanja Lange, Chloe Martindale, Lorenz Panny, and Joost Renes. CSIDH: an efficient post-quantum commutative group action. In Thomas Peyrin and Steven D. Galbraith, editors, *Advances in Cryptology - ASIACRYPT 2018 - 24th International Conference on the Theory and Application of Cryptology and Information Security, Brisbane, QLD, Australia, December 2-6, 2018, Proceedings, Part III*, volume 11274 of *Lecture Notes in Computer Science*, pages 395–427. Springer, 2018.
6. Wouter Castryck, Lorenz Panny, and Frederik Vercauteren. Rational isogenies from irrational endomorphisms. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology - EUROCRYPT 2020 - 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10-14, 2020, Proceedings, Part II*, volume 12106 of *Lecture Notes in Computer Science*, pages 523–548. Springer, 2020.
7. Jesús-Javier Chi-Domínguez and Francisco Rodríguez-Henríquez. Optimal strategies for CSIDH. *Adv. Math. Commun.*, 16(2):383–411, 2022.
8. Jesús-Javier Chi-Domínguez, Andre Esser, Sabrina Kunzweiler, and Alexander May. Low memory attacks on small key [csidh.
9. Craig Costello and Hüseyin Hisil. A simple and compact algorithm for SIDH with arbitrary degree isogenies. In Tsuyoshi Takagi and Thomas Peyrin, editors, *Advances in Cryptology - ASIACRYPT 2017 - 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part II*, volume 10625 of *Lecture Notes in Computer Science*, pages 303–329. Springer, 2017.
10. Craig Costello and Benjamin Smith. Montgomery curves and their arithmetic. *Journal of Cryptographic Engineering*, Mar 2017.
11. Jean-Marc Couveignes. Hard homogeneous spaces. Cryptology ePrint Archive, Paper 2006/291, 2006. `https://eprint.iacr.org/2006/291`.
12. Daniele Cozzo and Nigel P. Smart. Sashimi: Cutting up csi-fish secret keys to produce an actively secure distributed signing protocol. In Jintai Ding and Jean-

Pierre Tillich, editors, *Post-Quantum Cryptography*, pages 169–186, Cham, 2020. Springer International Publishing.

13. Luca De Feo. Mathematics of isogeny based cryptography. *CoRR*, abs/1711.04062, 2017.
14. Luca De Feo, Tako Boris Fouotsa, Péter Kutas, Antonin Leroux, Simon-Philipp Merz, Lorenz Panny, and Benjamin Wesolowski. SCALLOP: Scaling the CSI-FiSh. In Alexandra Boldyreva and Vladimir Kolesnikov, editors, *Public-Key Cryptography – PKC 2023*, pages 345–375, Cham, 2023. Springer Nature Switzerland.
15. Luca De Feo and Steven D. Galbraith. Seasign: Compact isogeny signatures from class group actions. In Yuval Ishai and Vincent Rijmen, editors, *Advances in Cryptology - EUROCRYPT 2019 - 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, May 19-23, 2019, Proceedings, Part III*, volume 11478 of *Lecture Notes in Computer Science*, pages 759–789. Springer, 2019.
16. Luca De Feo, Jean Kieffer, and Benjamin Smith. Towards practical key exchange from ordinary isogeny graphs. In Thomas Peyrin and Steven D. Galbraith, editors, *Advances in Cryptology - ASIACRYPT 2018 - 24th International Conference on the Theory and Application of Cryptology and Information Security, Brisbane, QLD, Australia, December 2-6, 2018, Proceedings, Part III*, volume 11274 of *Lecture Notes in Computer Science*, pages 365–394. Springer, 2018.
17. Luca De Feo, David Kohel, Antonin Leroux, Christophe Petit, and Benjamin Wesolowski. Sqisign: Compact post-quantum signatures from quaternions and isogenies. In Shiho Moriai and Huaxiong Wang, editors, *Advances in Cryptology - ASIACRYPT 2020 - 26th International Conference on the Theory and Application of Cryptology and Information Security, Daejeon, South Korea, December 7-11, 2020, Proceedings, Part I*, volume 12491 of *Lecture Notes in Computer Science*, pages 64–93. Springer, 2020.
18. Luca De Feo, David Kohel, Antonin Leroux, Christophe Petit, and Benjamin Wesolowski. SQISign: Compact post-quantum signatures from quaternions and isogenies. In Shiho Moriai and Huaxiong Wang, editors, *Advances in Cryptology - ASIACRYPT 2020 - 26th International Conference on the Theory and Application of Cryptology and Information Security, Daejeon, South Korea, December 7-11, 2020, Proceedings, Part I*, volume 12491 of *Lecture Notes in Computer Science*, pages 64–93. Springer, 2020.
19. Luca De Feo, Antonin Leroux, Patrick Longa, and Benjamin Wesolowski. New algorithms for the deuring correspondence - towards practical and secure sqisign signatures. In Carmit Hazay and Martijn Stam, editors, *Advances in Cryptology - EUROCRYPT 2023 - 42nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Lyon, France, April 23-27, 2023, Proceedings, Part V*, volume 14008 of *Lecture Notes in Computer Science*, pages 659–690. Springer, 2023.
20. David R. Kohel. *Endomorphism rings of elliptic curves over finite fields*. PhD thesis, University of California at Berkeley, 1996. `http://iml.univ-mrs.fr/~kohel/pub/thesis.pdf`.
21. Michael Meyer, Fabio Campos, and Steffen Reith. On lions and elligators: An efficient constant-time implementation of CSIDH. In Jintai Ding and Rainer Steinwandt, editors, *Post-Quantum Cryptography - 10th International Conference, PQCrypto 2019, Chongqing, China, May 8-10, 2019 Revised Selected Papers*, volume 11505 of *Lecture Notes in Computer Science*, pages 307–325. Springer, 2019.

22. Peter L Montgomery. Speeding the Pollard and elliptic curve methods of factorization. *Mathematics of computation*, 48(177):243–264, 1987.
23. Hiroshi Onuki, Yusuke Aikawa, Tsutomu Yamazaki, and Tsuyoshi Takagi. A constant-time algorithm of CSIDH keeping two points. *IEICE Trans. Fundam. Electron. Commun. Comput. Sci.*, 103-A(10):1174–1182, 2020.
24. Joost Renes. Computing Isogenies Between Montgomery Curves Using the Action of $(0, 0)$. In Tanja Lange and Rainer Steinwandt, editors, *Post-Quantum Cryptography*, pages 229–247, Cham, 2018. Springer International Publishing.
25. Alexander Rostovtsev and Anton Stolbunov. Public-key cryptosystem based on isogenies. Cryptology ePrint Archive, Paper 2006/145, 2006. `https://eprint.iacr.org/2006/145`.
26. Joseph H. Silverman. *The Arithmetic of Elliptic Curves*. Springer-Verlag, New York, NY, 2. aufl. edition, 2009.
27. Benjamin Smith. Pre- and post-quantum diffie-hellman from groups, actions, and isogenies. In Lilya Budaghyan and Francisco Rodríguez-Henríquez, editors, *Arithmetic of Finite Fields - 7th International Workshop, WAIFI 2018, Bergen, Norway, June 14-16, 2018, Revised Selected Papers*, volume 11321 of *Lecture Notes in Computer Science*, pages 3–40. Springer, 2018.
28. Anton Stolbunov. Reductionist security arguments for public-key cryptographic schemes based on group action. 2009.
29. Anton Stolbunov. Constructing public-key cryptographic schemes based on class group action on a set of isogenous elliptic curves. *Advances in Mathematics of Communications*, 4(2):215–235, 2010.
30. Jacques Vélu. Isogénies entre courbes elliptiques. *Comptes Rendus Hebdomadaires des Séances de l'Académie des Sciences, Série A*, 273:238–241, 7 1971.
31. Samuel S. Wagstaff, Jr. Pseudoprimes and a generalization of Artin's conjecture. *Acta Arithmetica*, 41:141–150, 1982.

A Computing a kernel generator

One task that poses a challenge is to find a point $G \in \mathcal{E}(\mathbb{F}_{q^k})$. In this section, we will illustrate an efficient method for computing a point with the necessary properties for use in the isogeny evaluation.

A.1 The subgroup H_k.

To compute a rational isogeny, our first step will be to sample a random point $P \in \mathcal{E}(\mathbb{F}_{q^k})$ of order ℓ. For this, letting $N_k := \#\mathcal{E}(\mathbb{F}_{q^k})$, one could sample a random point P, and compute $P_\ell = [N_k/\ell]P$. Then P_ℓ is either 0 or a point of order ℓ. If the order of P_ℓ is not ℓ, one tries again with a new choice of P.

Remark 1. In the special case that ℓ^2 divides $\#\mathcal{E}(\mathbb{F}_{q^k})$, we instead choose $N_k = exp(\mathbb{F}_{q^k})$, the exponent of the group order. We do this to avoid having a cofactor, N_k/ℓ, that "kills" certain torsion points.

In our context, we are assured that the P_ℓ we are looking for is *not* in $\mathcal{E}(\mathbb{F}_q)$, or indeed in any $\mathcal{E}(\mathbb{F}_{q^i})$, for any proper divisor i of k. We can therefore save some effort by sampling P_ℓ from the genuinely "new" subgroup of $\mathcal{E}(\mathbb{F}_{q^k})$.

Recall that $\mathcal{E}(\mathbb{F}_{q^i}) = \ker(\pi^i - [1])$ for each $i > 0$. For each $k > 0$, then we define an endomorphism

$$\eta_k := \Phi_k(\pi) \in \mathrm{End}(\mathcal{E})$$

where $\Phi_k(X)$ is the k-th cyclotomic polynomial (that is, the minimal polynomial over $\mathbb{Z}$ of the primitive k-th roots of unity in $\overline{\mathbb{F}}$). The subgroup

$$H_k := \ker(\eta_k) \subset \mathcal{E}(\mathbb{F}_{q^k})$$

satisfies

$$\mathcal{E}(\mathbb{F}_{q^k}) = H_k \oplus \sum_{i|k, i\neq k} \mathcal{E}(\mathbb{F}_{q^i}).$$

The key fact is that in our situation, $\mathcal{E}[\ell](\mathbb{F}_{q^k}) \subset H_k$.

Generating elements of $\mathcal{E}[\ell](\mathbb{F}_{q^k})$. We always have $\Phi_k(X) \mid X^k - 1$, so for each $k > 0$ there is an endomorphism

$$\delta_k := (\pi^k - [1])/\eta_k \in \mathrm{End}(\mathcal{E}),$$

and $\delta_k(\mathcal{E}(\mathbb{F}_{q^k})) \subset H_k$. We can therefore sample a point P_ℓ in $\mathcal{E}[\ell](\mathbb{F}_{q^k})$ by computing

$$P_\ell = [h_k/\ell]\delta_k(P) \qquad \text{where} \qquad h_k := \#H_k$$

and P is randomly sampled from $\mathcal{E}(\mathbb{F}_{q^k})$.

Table 6 lists the first few values of h_k and δ_k. We see that evaluating δ_k amounts to a few Frobenius operations (which are almost free, depending on the field representation) and a few applications of the group law, so this approach saves us a factor of at least $1/k$ in the loop length of the scalar multiplication (compared wih computing P_ℓ as $[N_k/\ell]P$), but for highly composite k we save much more.

The value $\varphi(k)$ of the Euler totient function plays an important role. We have $h_k = q^{\varphi(k)} + o(q^{\varphi(k)})$, so computing $[h_k/\ell]$ instead of $[N_k/\ell]$ allows us to reduce the loop length of basic scalar multiplication algorithms from $k \log_2 q$ to $\varphi(k) \log_2 q$, which is particularly advantageous when k is highly composite.

The action of Frobenius on H_k. The Frobenius endomorphism π commutes with η_k, and therefore restricts to an endomorphism of H_k. If $G \subset H_k$ is a subgroup of prime order ℓ and fixed by π, then π will act on G as multiplication by an integer eigenvalue λ (defined modulo ℓ). Since $\eta_k = \Phi_k(\pi) = 0$ on H_k by definition, we know that λ is a k-th root of unity in $\mathbb{Z}/\ell\mathbb{Z}$.

Scalar multiplication with Frobenius. Now, $H_k \cong \mathbb{Z}/d_k\mathbb{Z} \times \mathbb{Z}/e_k\mathbb{Z}$, where $d_k \mid e_k$ and (by the rationality of the Weil pairing) $d_k \mid q^k - 1$. Typically, d_k is very small compared with e_k. If $\ell \nmid d_k$, then we can replace H_k with the cyclic subgroup $H'_k := [d_k]H_k$, and h_k with $h'_k := e_k/d_k$. Now, π induces an endomorphism of H'_k, and therefore acts as multiplication by an eigenvalue λ defined modulo h'_k.

We want to compute $[c_k]P$ for P in H'_k, where $c_k := h'_k/\ell$. Since $\Phi_k(\pi) = 0$, the eigenvalue λ is a root of Φ_k (i.e., a primitive k-th root of unity) modulo h'_k. We can compute $a_0, \ldots, a_{k-1}$ such that

$$c_k \equiv \sum_{i=0}^{k-1} a_i \lambda^i \pmod{h'_k},$$

with each coefficient $a_i \approx (h'_k)^{1/\varphi(k)}$ in $O(q)$, and then

$$[c_k]P = \sum_{i=0}^{k-1} [a_i]\pi^i(P).$$

If we precompute the various sums of conjugates of P,then we can compute $[c_k]$ using a multiscalar multiplication algorithm with a loop of length only $\log_2 q$. This might be particularly interesting in the cases where $\varphi(k) = 2$ (which corresponds to GLV multiplication) or 4.

Table 6. The first few values of h_k and δ_k.

k	h_k	δ_k	$\varphi(k)$
1	1	$[1]$	
2	$N_2/N_1 = q + O(\sqrt{q})$	$\pi - [1]$	1
3	$N_3/N_1 = q^2 + O(q^{3/2})$	$\pi - [1]$	2
4	$N_4/N_2 = q^2 + O(q)$	$\pi^2 - [1]$	2
5	$N_5/N_1 = q^4 + O(q^{7/2})$	$\pi - [1]$	4
6	$(N_6N_1)/(N_2N_3) = q^2 + O(q^{3/2})$	$(\pi + [1])(\pi^3 - [1])$	2
7	$N_7/N_1 = q^6 + O(q^{7/2})$	$\pi - [1]$	6
8	$N_8/N_4 = q^4 + O(q^2)$	$\pi^4 - [1]$	4
9	$N_9/N_3 = q^6 + O(q^{4/2})$	$\pi^3 - [1]$	6
10	$(N_{10}N_1)/(N_2N_5) = q^4 + O(q^{7/2})$	$(\pi + [1])(\pi^5 - [1])$	4
11	$N_{11}/N_1 = q^{10} + O(q^{19/2})$	$\pi - [1]$	10
12	$(N_{12}N_2)/(N_4N_6) = q^4 + O(q^3)$	$(\pi^2 + [1])(\pi^6 - [1])$	4

Example 4. Consider $k = 3$: we have $\mathcal{E}(\mathbb{F}_{q^3}) \cong \mathcal{E}(\mathbb{F}_q) \oplus H_3$, and $\#H_3 = N_3/N_1$.

We first note that π^3 fixes the points in $\mathcal{E}(\mathbb{F}_{q^3})$, so $\pi^3 - [1] = [0]$ on $\mathcal{E}(\mathbb{F}_{q^3})$. By similar logic, the regular Frobenius map will fix the points in $\mathcal{E}(\mathbb{F}_q)$, meaning $\pi - [1] = [0]$ holds only for points contained entirely in the $\mathcal{E}(\mathbb{F}_q)$ portion of $\mathcal{E}(\mathbb{F}_{q^3})$. Therefore, by computing $P_H = (\pi - 1)P$, we are "killing" the $\mathcal{E}(\mathbb{F}_q)$ part of P, leaving only the part lying in the subgroup H_3. This computation is easy enough to do, and so now we need only compute $P_\ell = [N_3/N_1/\ell]P_H$, thereby saving us about a third of the multiplications.

The "twist trick". When k is even, if we use x-only scalar multiplication, then the following lemma allows us to work over $\mathbb{F}_{q^{k/2}}$ instead of $\mathbb{F}_{q^k}$. In the case $k = 2$, this is known as the "twist trick".

Lemma 1. *If k is even, then every point P in H_k has $x(P)$ in $\mathbb{F}_{q^{k/2}}$.*

Proof. If k is even, then η_k divides $\pi^{k/2}+1$, so $\pi^{k/2}$ acts as -1 on $H_k = \ker(\eta_k)$: that is, if P is in H_k, then $\pi^{k/2}(P) = -P$, so $x(P)$ is in $\mathbb{F}_{q^{k/2}}$.

Example 5. Consider $k = 6$. We take a random point R in $E(\mathbb{F}_{q^6})$, and compute $R' := \pi^3(R) - R$, then $P := \pi(R') + R'$; now $P = \delta_6(R)$ is in H_6, and $x(P)$ is in $\mathbb{F}_{q^3}$. We have $h_6 = N_1^2 - (q+1)N_1 + q^2 - q + 1 \approx q^2$, and we need to compute $x([c_{\ell,6}]P)$ where $c_{\ell,6} := h_6/\ell$. Since $x(P)$ is in $\mathbb{F}_{q^3}$, we can do this using x-only arithmetic and the Montgomery ladder working entirely over $\mathbb{F}_{q^3}$.

The improvements outlined in this section are summarized in Algorithm 7.

Algorithm 7: Computation of Kernel Generator.

Input: $\mathcal{E}$ an elliptic curve defined over $\mathbb{F}_q$, ℓ an integer, k such that $\mathbb{F}_{q^k}$ contains an ℓ-torsion point.
Output: P, a point on $\mathcal{E}(\mathbb{F}_{q^k})$ of order ℓ.

$P_\ell \leftarrow (0:1:0)$
repeat
  $P \leftarrow$ `RandomPoint`$(\mathcal{E}(\mathbb{F}_{q^k}))$
  $c \leftarrow h_k/\ell$ // h_k taken from Table 6.
  $P' \leftarrow \delta_k(P)$ // δ_k taken from Table 6.
  $P_\ell \leftarrow [c]P'$
until $P_\ell = (0:1:0)$
return P_ℓ