A Simple and Compact Algorithm for SIDH with Arbitrary Degree Isogenies

Craig Costello[1]([✉]) and Huseyin Hisil[2]

[1] Microsoft Research, Redmond, USA
craigco@microsoft.com
[2] Yasar University, Izmir, Turkey
huseyin.hisil@yasar.edu.tr

Abstract. We derive a new formula for computing arbitrary odd-degree isogenies between elliptic curves in Montgomery form. The formula lends itself to a simple and compact algorithm that can efficiently compute any low odd-degree isogenies inside the supersingular isogeny Diffie-Hellman (SIDH) key exchange protocol. Our implementation of this algorithm shows that, beyond the commonly used 3-isogenies, there is a moderate degradation in relative performance of $(2d+1)$-isogenies as d grows, but that larger values of d can now be used in practical SIDH implementations.

We further show that the proposed algorithm can be used to both compute isogenies of curves and evaluate isogenies at points, unifying the two main types of functions needed for isogeny-based public-key cryptography. Together, these results open the door for practical SIDH on a much wider class of curves, and allow for simplified SIDH implementations that only need to call one general-purpose function inside the fundamental computation of the large degree secret isogenies.

As an additional contribution, we also give new explicit formulas for 3- and 4-isogenies, and show that these give immediate speedups when substituted into pre-existing SIDH libraries.

Keywords: Post-quantum cryptography · Isogeny-based cryptography · SIDH · Montgomery curves

1 Introduction

Post-quantum Key Establishment. The existence of a quantum computer that is capable of implementing Shor's algorithm [36] at a large enough scale would have devastating consequences on the current public-key cryptographic standards and thus on the current state of cybersecurity [32]. Subsequently, the field of *post-quantum cryptography* (PQC) [4] is rapidly growing as cryptographers look for public-key solutions that can resist large-scale quantum adversaries. Recently, the USA's National Institute of Standards and Technology (NIST) began a process to develop new cryptographic standards and announced a call for PQC proposals with a deadline of November 30, 2017 [40].

Although the PQC community is currently examining alternatives to replace both traditional key establishment and traditional digital signature algorithms, there is an argument for scrutinising proposals in the former category with more haste than those in the latter. While digital signatures only need to be quantum-secure at the moment a powerful enough quantum adversary is realised, the realistic threat of long-term archival of sensitive data and a retroactive quantum break means that, ideally, key establishment protocols will offer quantum resistance long before such a quantum adversary exists [38].

Post-quantum key establishment proposals typically fall under one of three umbrellas:

(i) *Code-based.* Based on the McEliece cryptosystem [28] and its variants [28], modern proposals include Bernstein, Chou and Schwabe's McBits [5] and Misoczki *et al.*'s specialised MDPC-McEliece [10,29].

(ii) *Lattice-based.* Proposals here began with Hoffstein, Pipher and Silverman's standardised NTRUEncrypt [20], and in more recent times have been based on either Regev's learning with errors (LWE) problem [34] or Lyubashevsky, Peikert and Regev's ring variant (R-LWE) [27]. Peikert brought these problems to life in [33], and his protocols served as a basis for a number of recent implementations, including Bos *et al.*'s R-LWE key establishment software [7], Alkim *et al.*'s R-LWE successor NewHope [1], and Bos *et al.*'s LWE key establishment software Frodo [6].

(iii) *Isogeny-based.* Starting with the work of Couveignes [13] and with later work by Rostovsev and Stolbunov [35,39], Jao and De Feo proposed and implemented supersingular isogeny Diffie-Hellman (SIDH) key exchange [21]. In recent times a number of improvements and optimisations of their SIDH protocol have been proposed and implemented [2,11,12,15,26].

To date there is no clear frontrunner among the post-quantum key establishment proposals. In terms of functionality, all of the public implementations resulting from (i), (ii) and (iii) suffer the same drawback of requiring modifications (e.g., the Fujisaka-Okamoto transformation [17]) to achieve active security[1]. However, there are bandwidth versus performance trade-offs to consider when examining the above proposals; while SIDH affords significantly smaller public keys than its code- and lattice-based counterparts, the performance of the state-of-the-art SIDH software is currently orders of magnitude slower than the state-of-the-art implementations mentioned in (i) and (ii) above. The reason for this wide performance gap is that well-chosen code- and lattice-based instantiations typically involve simple matrix/vector operations over special, and comparatively tiny, implementation-friendly moduli that are either powers of 2 or very close to a power of 2. On the other hand, in addition to SIDH inheriting several of the more complex operations from traditional curve-based cryptography like scalar multiplications and pairings, it also involves a new style of isogeny arithmetic and requires a new breed of significantly larger underlying

---

[1] For (i), see [5, Sect. 6] and [23]; for (ii), see [33, Sect 5.3], [16] and [22]; for (iii), see [19,22].

finite fields. Whereas classical elliptic curve cryptography affords implementers the flexibility to cherry-pick the fastest underlying finite fields of sizes as small as 256 bits, most of the SIDH implementations to date have required extension fields of over one thousand bits whose underlying characteristic are of the form $p = 2^i 3^j - 1$. Imposing this special form of prime restricts both the number of SIDH-friendly fields available at a given security level and the number of field arithmetic optimisations possible for implementers.

Our Contributions. This paper presents a new algorithm for computing the fundamental operation in isogeny-based public-key cryptography, and in particular, within the SIDH protocol.

– *Odd-degree Montgomery isogenies.* We derive a new formula for odd-degree isogenies between Montgomery curves – see Theorem 1. Compared to Vélu's formulas for isogenies between Weierstrass curves, this formula is elegant and simple, both to write down and to implement. This formula immediately lends itself to a compact algorithm that computes arbitrary odd-degree isogenies.
– *Unifying the two isogeny operations.* SIDH operations require isogeny computations to be applied to elliptic curves within the isogeny class and to the points that lie on those curves. These two operations are typically different and require independent functions. For odd-degree isogenies, we show that both of these operations can be performed using the same core function by exploiting the simple connection between 2-torsion points and the Montgomery curve coefficient. This streamlines SIDH code, and for isogenies of degree 5 and above, has the added benefit of being significantly faster than performing the computations independently.
– *Simplified algorithm.* Together, the above two improvements culminate in a general-purpose algorithm that can efficiently compute isogenies of any odd degree. Coupled with specialised code for 2- and/or 4-isogenies, this allows arbitrary SIDH computations and gives rise to new possibilities within the SIDH framework. Our implementation benchmarks show that practitioners can lift the restriction of primes of the form $p = 2^i 3^j - 1$ without paying a huge performance penalty.
– *Faster 3- and 4-isogenies.* While the contributions mentioned above broaden the scope of curves that can be considered SIDH-friendly, they do not give an immediate speedup to existing SIDH implementations because the pre-existing formulas for 3-isogenies are a special case of Theorem 1. Nevertheless, as an auxiliary result, we give new dedicated 3- and 4-isogeny algorithms that do give immediate speedups. When plugging these new algorithms into Microsoft's recent v2.0 release of their SIDH library[2], Alice and Bob's key generations are both sped up by a factor 1.18x, while their shared secret computations are both sped up by a factor 1.11x.

Although this paper is largely geared towards SIDH key exchange, we note that almost all of the discussion applies analogously to other supersingular

---

[2] See https://github.com/Microsoft/PQCrypto-SIDH.

isogeny-based cryptographic schemes, e.g., to the other schemes proposed by
De Feo *et al.* [15], and to the recent isogeny-based signature scheme from Yoo
et al. [43].

Organisation. We give the preliminaries in Sect. 2. We provide the new formula
for odd-degree Montgomery isogenies in Sect. 3 and discuss its connection to
related works. We show how the point and curve isogeny computations can be
performed using the same function in Sect. 4, before presenting the general-
purpose odd-degree isogeny algorithm in Sect. 5. We provide implementation
benchmarks and conclude with some potential implications in Sect. 6. The faster
explicit formulas for 3- and 4-isogenies are presented in Appendix A.

Remark 1 (Even degree isogenies). Since any separable isogeny can be written
as a *chain* of prime degree isogenies [18, Theorem 25.1.2], our claim of treating
arbitrary degree isogenies on Montgomery curves follows from the coupling of
Theorem 1 (which covers isogenies of any odd degree) with the prior treatment
of 2-isogenies on Montgomery curves by De Feo *et al.* [15]. It is worth noting that
a technicality arises in the treatment of 2-isogenies on Montgomery curves: there
is currently no known way of computing a 2-isogeny directly from a generic 2-
torsion point without extracting a square root to transform the image curve into
Montgomery form. De Feo, Jao and Plût overcome this obstruction by making
use of a special 8-torsion point *lying above* the 2-torsion point in the kernel,
which is already available for use in the SIDH framework. In broader contexts,
however, the preservation of the Montgomery form under general 2-isogenies
might become problematic; in these cases even powers of 2 can be treated by the
application of 4-isogenies which do not need to compute square roots in order to
preserve the Montgomery form [15]. In Remark 2 we discuss the related work of
Moody and Shumow [31] on the (twisted) Edwards model [31]; in their case the 2-
isogeny formula also requires a square root computation to preserve the Edwards
form. Although Vélu's formulas for 2-isogenies between short Weierstrass curves
do not require square root computations, we believe it worthwhile to pose the
open question of finding efficient 2-isogenies that preserve either of the faster
Montgomery and/or twisted Edwards models (on input of a generic 2-torsion
point).

2   Preliminaries

Montgomery Curves. Unless stated otherwise, all elliptic curves E/K in this
paper are assumed to be written in Montgomery form [30]

$$E/K : by^2 = x^3 + ax^2 + x.$$

We will be dealing with the group of K-rational points on E, denoted $E(K)$,
which is the set of solutions $(x, y) \in K \times K$ to the above equation, furnished
with a point at infinity, $\mathcal{O}_E$. This point looks different under different projective

embeddings of E. The usual embedding into $\mathbb{P}^2$ via $x = X/Z$ and $y = Y/Z$ gives $\mathcal{O}_E = (0 : 1 : 0)$, but the proof of Theorem 1 makes use of the alternative embedding into $\mathbb{P}(1, 2, 1)$ via $x = X/Z$ and $y = Y/Z^2$, under which $\mathcal{O}_E = (1 : 0 : 0)$. The inverse of a point (x, y) is $(x, -y)$, and the number of points in $E(K)$ is always divisible by 4.

Let $P = (x_P, y_P)$ and $Q = (x_Q, y_Q)$ be such that $x_P \neq x_Q$. Then the coordinates of these points and the x-coordinates of their sum $P + Q$ and difference $P - Q$ are related by Montgomery's group law identities [30, p. 261]

$$x_{P+Q}(x_P - x_Q)^2 x_P x_Q = b(x_P y_Q - x_Q y_P)^2, \quad \text{and}$$
$$x_{P-Q}(x_P - x_Q)^2 x_P x_Q = b(x_P y_Q + x_Q y_P)^2. \tag{1}$$

Montgomery multiplies these equations to produce his celebrated differential arithmetic formulas [30, p. 262]

$$x_{P+Q} x_{P-Q} = (x_P x_Q - 1)^2 / (x_P - x_Q)^2, \quad \text{and}$$
$$x_{[2]P} = (x_P^2 - 1)^2 / (4x_P(x_P^2 + ax_P + 1)). \tag{2}$$

Assuming the usual embedding of E into $\mathbb{P}^2$, then following [30], we use $\mathbf{x}$ throughout to denote the subsequent projection of points into $\mathbb{P}^1$ that *drops* the Y-coordinate, i.e.,

$$\mathbf{x} : \quad E \setminus \{\mathcal{O}_E\} \to \mathbb{P}^1, \qquad (X : Y : Z) \mapsto (X : Z).$$

Applying this to (2) gives Montgomery's two algorithms for differential arithmetic in $\mathbb{P}^1$, i.e.,

$$\texttt{xDBL} : \quad (\mathbf{x}(P), a) \mapsto \mathbf{x}([2]P), \quad \text{and}$$
$$\texttt{xADD} : \quad (\mathbf{x}(P), \mathbf{x}(Q), \mathbf{x}(P - Q)) \mapsto \mathbf{x}(P + Q). \tag{3}$$

If ℓ is odd, then the ℓ-th division polynomial of an elliptic curve E/K is written as $\psi_\ell(x) \in K[x]$, and this vanishes precisely at the nontrivial ℓ-torsion points, i.e., the points P such that $[\ell]P = \mathcal{O}_E$. The first two nontrivial odd division polynomials on the Montgomery curve $E/K : by^2 = x^3 + ax^2 + x$ are

$$\psi_3(x) = 3x^4 + 4ax^3 + 6x^2 - 1,$$
$$\psi_5(x) = 5x^{12} + 20ax^{11} + (16a^2 + 62)x^{10} + 80ax^9 - 105x^8 - 360ax^7$$
$$- 60(5 + 4a^2)x^6 - 16a(23 + 4a^2)x^5 - 5(25 + 32a^2)x^4 - 140ax^3 - 50x^2 + 1. \tag{4}$$

SIDH. Let $p = f \cdot n_A n_B \pm 1$ be a large prime where $\gcd(n_A, n_B) = 1$ and f is a small cofactor. SIDH [21] works in the isogeny class of supersingular elliptic curves over $\mathbb{F}_{p^2}$, all of which have cardinality $(p \mp 1)^2 = (f \cdot n_A n_B)^2$. Let E be a public starting curve in this isogeny class. To generate her public key, Alice chooses a secret subgroup G_A of order n_A on E and computes her public key as E/G_A. Likewise, Bob chooses a secret subgroup G_B of order n_B and computes his public key as E/G_B. The shared secret is then $E/\langle G_A, G_B \rangle$,

and so long as computing this from E, E/G_A and E/G_B is hard, this offers an alternative instantiation of the Diffie-Hellman protocol [14]. The key to the SIDH construction is ensuring that both parties exchange enough information to allow the mutual computation of $E/\langle G_A, G_B\rangle$, while still hiding their secret keys.

To achieve this, Jao and De Feo [21] propose that the public keys also contain the images of certain public points under the isogenies defined by their secret subgroups. If $\phi_A : E \to E/G_A$ is the secret isogeny corresponding to the subgroup G_A, then Alice not only sends Bob the curve E/G_A, but also the image of ϕ_A on two points P_B and Q_B whose linear combinations generate the set of subgroups chosen by Bob, i.e., Alice's public key is $\mathrm{PK}_A = (E/G_A, \phi_A(P_B), \phi_A(Q_B))$. Similarly, if linear combinations of P_A and Q_A generate the set of subgroups chosen by Alice, then Bob's public key is $\mathrm{PK}_B = (E/G_B, \phi_B(P_A), \phi_B(Q_A))$. In this way Alice's key generation amounts to randomly choosing two secret integers $u_A, v_A \in \mathbb{Z}_{n_A}$, computing $G_A = \langle [u_A]P_A + [v_A]Q_A\rangle$, and upon receipt of Bob's public key, she can then compute $E/\langle G_A, G_B\rangle = (E/G_B)/\langle [u_A]\phi_B(P_A) + [v_A]\phi_B(Q_A)\rangle$. Bob proceeds analogously, and both parties compute the shared secret as the j-invariant of $E/\langle G_A, G_B\rangle$.

In order for SIDH to be secure, n_A and n_B must be exponentially large so that Alice and Bob have an exponentially large keyspace. On the other hand, in order for SIDH to be practical, the computation of the n_A- and n_B-isogenies must be manageable. To achieve this, Jao and De Feo propose that $n_A = \ell_A^{e_A}$ and $n_B = \ell_B^{e_B}$ for ℓ_A and ℓ_B small; in this way there are $\ell_A^{e_A-1}(\ell_A + 1)$ secret cyclic subgroups of order n_A for Alice to choose from, and her secret isogeny computations can be performed as the composition of e_A low-degree ℓ_A-isogenies (the analogous statement applies to Bob). In all of the SIDH implementations to date [2,11,12,15,26], $\ell_A = 2$ and $\ell_B = 3$, and Alice computes her 2^{e_A}-isogeny as a composition of 2- and/or 4-isogenies (see [12,15]), while Bob computes his 3^{e_B}-isogeny as a composition of 3-isogenies. One consequence of this paper is to facilitate practical ℓ^e-isogenies where $\ell \geq 5$.

Following [21, Fig. 2], one way to compute an ℓ^e cyclic isogeny ϕ on the curve E_0 is to start with a point P_0 of order ℓ^e, compute the point $[\ell^{e-1}]P_0$ of order ℓ, then use Vélu's formulas [41] to compute the ℓ-isogeny $\phi_0 : E_0 \to E_1$ with $\ker(\phi_0) = \langle [\ell^{e-1}]P_0\rangle$, and evaluate it at P_0 to give $\phi_0(P_0) = P_1 \in E_1$. Note that pushing P_0 through the ℓ-isogeny reduces the order of its image point, P_1, by a factor of ℓ on E_1. This process is then repeated at each new iteration by first computing the order ℓ point $[\ell^{e-1-i}]P_i$, then the ℓ-isogeny $\phi_i : E_i \to E_{i+1}$, and finally the computation of the new point $P_{i+1} = \phi_i(P_i)$; this is done until $i = e-1$ and we have the final curve $E_e = \phi_{e-1} \circ \cdots \circ \phi_0(E_0) = \phi(E_0)$.

In their extended article, De Feo *et al.* [15] detailed a much faster approach towards the computation described above. Roughly speaking, they achieve large speedups by storing intermediate multiples of the P_i at each step and evaluating ϕ_i at these multiples in such a way that the length of the scalar multiplication to find an order-ℓ point on E_{i+1} is reduced. They aim to minimise the overall cost of

the ℓ^e-isogeny computation by comparing the costs of point multiplications and isogeny computations and studying the optimisation problem in a combinatorial context – see [15, Sect. 4.2.2].

Following [12], in order to thwart simple timing attacks [24], the fastest way to compute SIDH operations in a constant-time fashion is to (i) perform point operations on the projective line $\mathbb{P}^1$ associated to Montgomery's x-coordinate, i.e., using the map $\mathbf{x}$ above, and (ii) to also perform isogeny operations projectively in $\mathbb{P}^1$ by ignoring the b coefficient (in the same way the Y-coordinate is ignored in the point arithmetic). The reasoning here is that the j-invariant (i.e., the isomorphism class) of the Montgomery curve $E/K : by^2 = x^3 + ax^2 + x$ is $j(E) = 256(a^2 - 3)^3/(a^2 - 4)$, which is independent of b, as is the differential arithmetic arising from (3). All of the formulas and algorithms we describe in the remainder of this paper fit into this same framework.

3   Coordinate Maps for Odd-Degree Montgomery Isogenies

At the heart of this paper is the coordinate maps in Eq. (6) of Theorem 1 below. Although we are mostly concerned with the SIDH-specific applications to come in the following sections, we believe that the simplicity and usability of the formula may be of interest outside the realm of SIDH, so we leave the underlying field unspecified and state the isogeny formula in full. We follow the theorem with a discussion of the related work of Moody and Shumow [31].

Theorem 1. *For a field K with $\mathrm{char}(K) \neq 2$, let $P \in E(\bar{K})$ be a point of order $\ell = 2d + 1$ on the Montgomery curve $E/K : by^2 = x^3 + ax^2 + x$ and write $\sigma = \sum_{i=1}^{d} x_{[i]P}$, $\tilde{\sigma} = \sum_{i=1}^{d} 1/x_{[i]P}$ and $\pi = \prod_{i=1}^{d} x_{[i]P}$. The Montgomery curve*

$$E'/K : b'y^2 = x^3 + a'x^2 + x \tag{5}$$

with

$$a' = (6\tilde{\sigma} - 6\sigma + a) \cdot \pi^2 \qquad \text{and} \qquad b' = b \cdot \pi^2$$

is the codomain of the ℓ-isogeny $\phi : E \to E'$ with $\ker(\phi) = \langle P \rangle$, which is defined by the coordinate maps

$$\phi : (x, y) \mapsto (f(x), y \cdot f'(x)), \tag{6}$$

where

$$f(x) = x \cdot \prod_{i=1}^{d} \left(\frac{x \cdot x_{[i]P} - 1}{x - x_{[i]P}} \right)^2,$$

and $f'(x)$ is its derivative.

Proof. The proof follows along the lines of Washington's proof of Vélu's formulas on general Weiestrass curves [42, Theorem 12.16]. Write

$$\phi : (x, y) \mapsto (X, Y),$$

where $X = f(x) = x \cdot u(x)^2/w(x)^2$ and $Y = y \cdot f'(x)$, with $u(x) = \prod_{i=1}^{d}(x \cdot x_i - 1)$ and $w(x) = \prod_{i=1}^{d}(x - x_i)$, and write $G = \langle P \rangle$. Since X and Y are rational functions of x and y, they are functions on E, and it is clear that the only poles of X and Y are at the points in G. Our main goal is to show that the function

$$F(X, Y) = b'Y^2 - (X^3 + a'X^2 + X) \tag{7}$$

is 0. The idea is to introduce a uniformising parameter t at $\mathcal{O}_E$ and ultimately show that $F(X, Y) \in O(t)$, i.e., that $F(X, Y)$ vanishes at $\mathcal{O}_E$. Now, since $x_{[i]P} = x_{[\ell-i]P}$ for $i = 1 \ldots d$, it follows from (2) that for any $Q = (x_Q, y_Q) \notin G$,

$$f(x_Q) = \prod_{T \in G} x_{Q+T},$$

and therefore that the functions X and Y are invariant under translation by elements of G. Thus, if we can show that $F(X, Y)$ vanishes at $\mathcal{O}_E$, we will have also shown that it vanishes at all of the points in G. Since the function $F(X, Y)$ can only have poles at the points in G, the only possibility is that $F(X, Y)$ has no poles, which means that it is constant [42, Proposition 11.1(3)]. Furthermore, since it is 0 at infinity, it must be that $F(X, Y)$ is identically zero, and therefore that X and Y satisfy the Montgomery curve equation in (5).

To show that $F(X, Y)$ vanishes at $\mathcal{O}_E$, let $t = x/y$ be a uniformising parameter at $\mathcal{O}_E$ and let $s = 1/y$. Dividing $by^2 = x^3 + ax^2 + x$ by y^3 and rearranging yields

$$s = (t^3 + at^2s + ts^2)/b.$$

Continually substituting this value for s into the above equation eventually yields

$$s = t_3 \cdot t^3 + t_5 \cdot t^5 + t_7 \cdot t^7 + t_9 \cdot t^9 + \ldots,$$

where

$$t_3 = 1/b, \qquad t_5 = a/b^2, \qquad t_7 = (1 + a^2)/b^3, \qquad t_9 = a(3 + a^2)/b^4,$$
$$t_{11} = (6a^2 + a^4 + 2)/b^5, \qquad t_{13} = a(a^4 + 10a^2 + 10)/b^6.$$

Since $y = 1/s$ and $x = ty$, we invert the above equation to give

$$y = bt^{-3} \cdot (y_0 + y_2 \cdot t^2 + y_4 \cdot t^4 + y_6 \cdot t^6 + \ldots), \quad \text{and}$$
$$x = bt^{-2} \cdot (y_0 + y_2 \cdot t^2 + y_4 \cdot t^4 + y_6 \cdot t^6 + \ldots) \tag{8}$$

where

$$y_0 = 1, \qquad y_2 = -a/b, \qquad y_4 = -1/b^2, \qquad y_6 = -a/b^3,$$
$$y_8 = -(a^2 + 1)/b^4, \qquad y_{10} = -(a^3 + 3a)/b^5. \qquad y_{12} = (a^6 + 14a^4 + 24a^2 + 3)/b^6.$$

From (6), we write

$$X = x \cdot f_1(x)^2 \cdot f_2(x)^2 \cdot \cdots \cdot f_d(x)^2, \tag{9}$$

where

$$f_i(x) = \frac{x_{[i]P} \cdot x - 1}{x - x_{[i]P}},$$

for $1 \leq i \leq d$. Substitution of (8) gives

$$f_i(t) = \frac{x_{[i]P} \cdot \left(bt^{-2} \cdot (y_0 + y_2 \cdot t^2 + y_4 \cdot t^4 + y_6 \cdot t^6 + \dots)\right) - 1}{(bt^{-2} \cdot (y_0 + y_2 \cdot t^2 + y_4 \cdot t^4 + y_6 \cdot t^6 + \dots)) - x_{[i]P}},$$

$$= x_{[i]P} + \left[\frac{x_{[i]P}^2 - 1}{b}\right] \cdot t^2 + \left[\frac{(x_{[i]P}^2 - 1)(a + x_{[i]P})}{b^2}\right] \cdot t^4$$

$$+ \left[\frac{(x_{[i]P}^2 - 1)((a + x_{[i]P})^2 + 1)}{b^3}\right] \cdot t^6 \quad + \quad O(t^8).$$

Squaring the above equation yields

$$f_i(t)^2 = x_{[i]P}^2 + \left[\frac{2x_{[i]P}(x_{[i]P}^2 - 1)}{b}\right] \cdot t^2 + \left[\frac{(x_{[i]P}^2 - 1)(3x_{[i]P}^2 + 2ax_{[i]P} - 1)}{b^2}\right] \cdot t^4$$

$$+ \left[\frac{2(x_{[i]P}^2 - 1)(a^2 x_{[i]P} + 3ax_{[i]P}^2 + 2x_{[i]P}^3 - a)}{b^3}\right] \cdot t^6 \quad + \quad O(t^8). \tag{10}$$

Substitution of (8) and (10) into (9) gives

$$X(t) = X_{-2} \cdot t^{-2} + X_0 + X_2 \cdot t^2 + X_4 \cdot t^4 + O(t^6). \tag{11}$$

where

$$X_{-2} = b\pi^2$$
$$X_0 = -\pi^2(2(\tilde{\sigma} - \sigma) + a)$$
$$X_2 = -\frac{4\pi^4((\sigma - \tilde{\sigma})(a + 3(\tilde{\sigma} - \sigma)) + 1) + 1}{5b\pi^2}$$
$$X_4 = -\frac{12\pi^4(\sigma - \tilde{\sigma})a^2 + (3 - 16\pi^4((\sigma - \tilde{\sigma})^2 - 2))a - 10(\sigma - \tilde{\sigma})(8\pi^4(\sigma - \tilde{\sigma})^2 - 1)}{35b^2\pi^2}.$$

Now, the product rule gives $X'(x) = X'(t) \cdot (x'(t))^{-1}$, so from (11) we have

$$X'(t) = t^{-3} \cdot \left(-2X_{-2} + 2X_2 t^4 + 4X_4 t^6 + O(t^8)\right), \tag{12}$$

and from (8) we have

$$x'(t) = bt^{-3} \cdot (-2y_0 + 2y_4 \cdot t^4 + 4y_6 \cdot t^6 + 6y_8 \cdot t^8 \dots).$$

Inverting the above equation yields

$$x'(t)^{-1} = -\frac{t^3}{2by_0^2} \cdot \left(y_0 + y_4 \cdot t^4 + 2y_6 \cdot t^6 + \left[\frac{3y_0y_8 + y_4^2}{y_0}\right] \cdot t^8\right.$$
$$\left. + \left[\frac{4y_0y_{10} + 4y_4y_6}{y_0}\right] \cdot t^{10} + O(t^{12})\right). \qquad (13)$$

We now have all the ingredients to write $F(X, Y)$ entirely in terms of t. We write

$$F(t) = b' \left(y(t) \cdot X'(t) \cdot x'(t)^{-1}\right)^2 - \left(X(t)^3 + a'X(t)^2 + X(t)\right),$$

Substituting (8), (11), (12) and (13) into the above equation, and collecting coefficients, yields

$$F(X, Y) = F_{-6} \cdot t^{-6} + F_{-4} \cdot t^{-4} + F_{-2} \cdot t^{-2} + F_0 + O(t), \qquad (14)$$

where

$$F_{-6} = X_{-2}^2 \cdot (b' - X_{-2}),$$
$$F_{-4} = -X_{-2}^2 \cdot \frac{2ab' + b(a' + 3X_0)}{b},$$
$$F_{-2} = X_{-2} \cdot \frac{((a^2 - 4)X_{-2} - 2b^2X_2) \cdot b' - b^2(2a'X_0 + 3X_0^2 + 3X_2X_{-2} + 1)}{b^2},$$
$$F_0 = \frac{4X_{-2}(aX_2 - bX_4)b' - b \cdot (a'X_0^2 + 2a'X_2X_{-2} + X_0^3 + 6X_{-2}X_0X_2 + 3X_4X_{-2}^2 + X_0)}{b}.$$

With X_{-2}, X_0, X_2 and X_4 as in (11), and with a' and b' from (5), we get $F_{-6} = F_{-4} = F_{-2} = F_0 = 0$. Thus, we have $F(X, Y) \in O(t)$, which means that $F(X, Y)$ vanishes at $\mathcal{O}_E$, and thus (as detailed above) that $F(X, Y)$ is identically zero. It follows that X and Y satisfy the equation for E' in (5), and thus that ϕ is a rational map from E to E'. Since E is a smooth curve, we have that ϕ is a morphism [37, Proposition II.2.1]. To show that ϕ is an isogeny, we project into $\mathbb{P}(1, 2, 1)$, where $\mathcal{O}_E = (1 \colon 0 \colon 0)$. Substitution of $x = X/Z$ and $y = Y/Z^2$ into (6) reveals that $\phi(\mathcal{O}_E) = (1 \colon 0 \colon 0) = \mathcal{O}_{E'}$, and we have established that ϕ is an isogeny [37, III.4]. This completes the proof.   □

In comparison to Vélu's formulas [41] on general Weierstrass curves, the simplicity of Eq. (6) lies in the fact that it factors neatly across the different multiples of P. This lends itself to the simple algorithm we describe in Sect. 5.

Remark 2. To our knowledge, the work of Moody and Shumow [31] is the only prior work to investigate arbitrary degree isogenies on non-Weierstrass models. They managed to successfully derive general isogenies on both (twisted) Edwards curves [31, Theorem 3] and on Huff curves [31, Theorem 5] without passing back and forth to Vélu's formulas on Weierstrass models. Given the 'uniform-variable' formulas in [31, Sect. 4.4], we could have presumably arrived at (6) by exploiting the birational equivalence between twisted Edwards and Montgomery curves [3, Theorem 3.2]. In particular, there is a simple relationship between the

twisted Edwards y-coordinate and the Montgomery x-coordinate, and subsequently, there are Edwards y-only analogues of Montgomery's x-only differential arithmetic that offer favourable trade-offs in certain ECC scenarios[3]. However, our experiments seemed to suggest that these trade-offs evaporate in SIDH when the curve constants are treated projectively. Nevertheless, given the similarities between the y-only isogeny formula in [31, Theorem 4] and our Theorem 1, it could be that there are savings to be gained in a twisted Edwards version of SIDH, or perhaps in some sort of hybrid that passes back and forth between the two models – see [9]. We leave this investigation open, pointing out that the sorts of trade-offs discussed in [9] can become especially favourable in SIDH, due to the large field sizes and the nature of arithmetic in quadratic extension fields.

We conclude this section by pointing out that (in our case) it is a simple exercise to transform Eq. (6) into an analogue that, rather than writing the isogeny map in terms of the coordinates of the torsion points à la Vélu, instead writes it in terms of the coefficients of the polynomial defining the kernel subgroup à la Kohel [25, Sect. 2.4]. While a Kohel-style formulation of our formula is arguably more natural from a mathematical perspective, the way it is factored and written in (6) is more natural from an algorithmic perspective.

4   Computing the Isogenous Curve Using the 2-Torsion

Let $\phi : E \mapsto E'$ be the isogeny of Montgomery curves in Theorem 1 and let $Q \in E$ be any point where $Q \notin \ker(\phi)$. All supersingular isogeny-based cryptosystems, and in particular all known implementations of SIDH [2,11,12,15,26], require separate functions for computing isogenous curves, i.e., $\mathtt{iso\_curve} : E \mapsto \phi(E)$, and for evaluating the isogeny at points, i.e., $\mathtt{iso\_point} : Q \mapsto \phi(Q)$. In this brief section we show that these two functions can be unified in the computation of odd-degree isogenies. The idea is to exploit the correspondence between the 2-torsion points and the curve-twist isomorphism class, and to replace calls to the $\mathtt{iso\_curve}$ function with calls to $\mathtt{iso\_point}$ on the input of 2-torsion points. Pushing 2-torsion points through an odd-degree isogeny preserves their order on the image curve, and so the correspondence between 2-torsion points and the isogenous curves they lie on remains an invariant throughout the SIDH algorithm.

On the Montgomery curve $E/K : y^2 = x^3 + ax^2 + x$, the three affine points of order 2 in $E(\bar{K})$ are

$$P_0 = (0,0), \qquad P_\alpha = (\alpha, 0), \qquad \text{and} \qquad P_{1/\alpha} = (1/\alpha, 0),$$

where $a = -(\alpha^2 + 1)/\alpha$. Note that the full 2-torsion is K-rational if $x^2 + ax + 1$ is reducible in $K[x]$, i.e., if $\alpha \in K$; this is typically the case in SIDH and is therefore assumed in this section.

---

[3] See http://hyperelliptic.org/EFD/g1p/auto-edwards-yz.html.

Under the **x** map from Sect. 2, the 2-torsion points are then

$$\mathbf{x}(P_0) = (0:1), \qquad \mathbf{x}(P_\alpha) = (X_\alpha : Z_\alpha), \qquad \text{and} \qquad \mathbf{x}(P_{1/\alpha}) = (Z_\alpha : X_\alpha),$$

and in $\mathbb{P}^1$ we now have

$$(a : 1) = (X_\alpha^2 + Z_\alpha^2 : -X_\alpha Z_\alpha). \tag{15}$$

Observe that for the isogeny ϕ described in Theorem 1, the point $P_0 = (0,0) \in E$ is mapped to the point $\phi(P_0) = (0,0) \in E'$. Since E' is a Montgomery curve and 2-torsion points preserve their order under odd isogenies, it must be that

$$\mathbf{x}(\phi(P_0)) = (0:1), \qquad \mathbf{x}(\phi(P_\alpha)) = (X_\alpha' : Z_\alpha'), \qquad \text{and} \qquad \mathbf{x}(\phi(P_{1/\alpha})) = (Z_\alpha' : X_\alpha'),$$

so that the relation in (15) between 2-torsion coordinates and the curve coefficient holds on the new curve.

Rather than thinking of the Montgomery curve as being represented by the coefficient $(a : 1) = (A : C)$, we can (without loss of generality) think of it as being represented by the 2-torsion point $(X_\alpha : Z_\alpha)$. A close inspection of Theorem 1 reveals that, for values of d greater than 3, computing the isogenous curve via (5) becomes increasingly more expensive than passing a 2-torsion point through (6). In these cases a function for computing (5) is no longer needed. If, during the current iteration, the curve constant $(A : C)$ is needed for point operations (e.g., the multiplication-by-ℓ map), then we can recover A using (15) at a cost[4] of $2\mathbf{S} + 5\mathbf{a}$. In fact, the general multiplication-by-ℓ routine is the Montgomery ladder [30] that calls `xDBL` as a subroutine, and (in SIDH) `xDBL` makes use of the constant $(a - 2 : 4) = ((A - 2C)/4 : C)$. Parsing directly to this format is slightly faster than parsing to $(A : C)$, since from (15) we have $((A - 2C)/4 : C) = ((X_\alpha + Z_\alpha)^2 : (X_\alpha - Z_\alpha)^2 - (X_\alpha + Z_\alpha)^2)$, which can be computed in $2\mathbf{S} + 3\mathbf{a}$.

Although parsing from α to $(a : 1) = (1 + \alpha^2 : -\alpha)$ is trivial, parsing in the other direction requires a square root computation in general. We never have to do this, however, since (i) during key generation, the starting curve is fixed and so the corresponding 2-torsion point(s) can be thought of as system parameters[5], and (ii) for the subsequent shared secret computations, we can happily replace a with α as the description of the supersingular curve in the (compressed or uncompressed) public key[6]. In the next section we use the notation `a_from_alpha` to represent the function that performs this cheap parsing.

---

[4] As usual, we use $\mathbf{M}$, $\mathbf{S}$, $\mathbf{a}$ and $\mathbf{I}$ to denote the costs of multiplications, squarings, additions/subtractions and inversions in the field $\mathbb{F}_{p^2}$.

[5] All public implementations of SIDH currently take the starting curve to be $E/\mathbb{F}_{p^2} : y^2 = x^3 + x$, where $\mathbb{F}_{p^2} = \mathbb{F}_p(i)$ with $i^2 + 1$. In these scenarios the starting 2-torsion point can be defined by setting $\alpha = i$.

[6] We note that the uncompressed keys in the SIDH library associated with [12] do not send the curve constant a explicitly, so would require modest modifications to take advantage of this 2-torsion technique. The compressed key format from the subsequent work in [11] does send a in the public key so has immediately compatibility.

Remark 3. If P and $Q \neq \pm P$ are two points on the Montgomery curve $E/K : by^2 = x^3 + ax^2 + x$, then the curve constant a relates to their x-coordinates and the x-coordinate of their difference via [12, Remark 4]

$$a = \frac{(1 - x_P x_Q - x_P x_{Q-P} - x_Q x_{Q-P})^2}{4 x_P x_Q x_{Q-P}} - x_P - x_Q - x_{Q-P}. \qquad (16)$$

Thus, if we ever have three points on an isogenous curve whose coefficient has not been computed, we can use the projective version of the above equation to recover $(a : 1) = (A : C)$ from $\mathbf{x}(P)$, $\mathbf{x}(Q)$ and $\mathbf{x}(Q-P)$. Since the cost of computing the isogenous curve using the 2-torsion technique grows as the degree of the isogeny grows, while the cost of computing the isogenous curve via (16) is fixed, there will obviously be a crossover point when taking advantage of three available points becomes faster. Based on the cost of computing one ℓ-isogeny presented in the next section, and on the projective version of (16) costing $8\mathbf{M} + 5\mathbf{S} + 11\mathbf{a}$, it will be faster to use the above after $d \geq 2$. Following [12], we note that there is always three such x-coordinates that can be exploited during key generation, namely the three x-coordinates whose image under the isogeny forms (part of) the public key. During the shared secret computation, however, there will not always be three points available at each stage. Thus, we recommend that unless d is very large (so that the performance benefits of using (16) over an additional isogeny evaluation will be visible), it is most simple to stick to the 2-torsion approach in this section through the SIDH algorithm.

5   A General-Purpose Algorithm for Arbitrary Odd-Degree Isogenies

We now turn to deriving an optimised algorithm for arbitrary odd-degree isogeny evaluation based on Theorem 1. Since we are working exclusively within the $\mathbb{P}^1$ framework under the map $\mathbf{x}$, the only equation we need to recall is (6), which we rewrite as

$$f(x) = x \cdot \left(\prod_{i=1}^{d} \left(\frac{x \cdot x_{[i]P} - 1}{x - x_{[i]P}} \right) \right)^2.$$

We begin working this into an algorithm by first projectivising into $\mathbb{P}^1$, writing $(X_i : Z_i) = (x_{[i]P} : 1)$ for $i = 1 \ldots d$, $(X : Z) = (x : 1)$ for the indeterminate coordinate where the isogeny is evaluated, and $(X' : Z') = \mathbf{x}(\phi(x, y))$ for the result. Then

$$X' = X \cdot \left(\prod_{i=1}^{d} (X \cdot X_i - Z_i \cdot Z) \right)^2, \quad \text{and}$$

$$Z' = Z \cdot \left(\prod_{i=1}^{d} (X \cdot Z_i - X_i \cdot Z) \right)^2$$

At first glance it appears that computing the pairs $(X \cdot X_i - Z_i \cdot Z)$ and $(X \cdot Z_i - X_i \cdot Z)$ will cost $4\mathbf{M} + 2\mathbf{a}$ each, but following Montgomery [30], we can achieve this in $2\mathbf{M} + 6\mathbf{a}$ by rewriting the above as

$$X' = X \cdot \Big(\prod_{i=1}^{d} \Big[(X - Z)(X_i + Z_i) + (X + Z)(X_i - Z_i) \Big] \Big)^2, \quad \text{and}$$

$$Z' = Z \cdot \Big(\prod_{i=1}^{d} \Big[(X - Z)(X_i + Z_i) - (X + Z)(X_i - Z_i) \Big] \Big)^2. \tag{17}$$

Observe that when $d > 1$ the values of $X - Z$ and $X + Z$ can be reused across the d expressions in both of the products above. Furthermore, the isogeny ϕ is usually going to be evaluated at multiple *points* of the form $(X : Z)$, and this will always be the case if the 2-torsion technique from the previous section is employed. Thus, suppose the isogeny is to be evaluated at the n elements $(\mathbf{X}_1 : \mathbf{Z}_1), \ldots, (\mathbf{X}_n : \mathbf{Z}_n)$, where we use boldface to distinguish these points and the coordinates of the i-th multiples of the kernel generator P. We note at once that the values of $(X_i + Z_i)$ and $(X_i - Z_i)$ can now also be reused across the n elements evaluated by the isogeny. This mutual recycling across both sets of points suggests a simple subroutine that merely computes the sum and difference of these pairwise products as in (17): we dub this routine `CrissCross` and present it in Algorithm 1 for completeness.

Algorithm 1. `CrissCross`: $K^4 \to K^2$.

Input: $(\alpha, \beta, \gamma, \delta) \in K^4$
Output: $(\alpha\delta + \beta\gamma, \alpha\delta - \beta\gamma) \in K^2$
Cost: $2\mathbf{M} + 2\mathbf{a}$.
1 $(\mathsf{t}_1, \mathsf{t}_2) \leftarrow (\alpha \cdot \delta, \beta \cdot \gamma)$ // $2\mathbf{M}$
2 **return** $(t_1 + t_2, t_1 - t_2)$ // $2\mathbf{a}$

Now, on input of the kernel generator $\mathbf{x}(P) = (X_1 : Z_1)$, the first step of the main algorithm will be to generate the $d - 1$ additional elements $\mathbf{x}([i]P) = (X_i : Z_i)$. This subroutine is called `KernelPoints` and we present it in Algorithm 2. Since it must start with a call to `xDBL`[7], we also need to input the modified curve constant $(\hat{A} : \hat{C}) = (a - 2 : 4)$.

Looking back at (17), we can see that once the $(X_i : Z_i)$ have been computed for $i = 1, \ldots, d$, they can immediately be overwritten by their sum and difference pairs through assigning $(\hat{X}_i, \hat{Z}_i) \leftarrow (X_i + Z_i, X_i - Z_i)$ in preparation for `CrissCross`. Based on (17), we now present an algorithm for evaluating a

---

[7] For many values of d, all of the kernel elements can be generated by repeated calls to `xDBL`, which is slightly cheaper than `xADD` in our context. However, for the sake of general applicability, we make repeated calls to `xADD` after the initial `xDBL` in `KernelPoints`.

Algorithm 2. KernelPoints: $\mathbb{P}^1 \times \mathbb{P}^1 \to (\mathbb{P}^1)^d$.

    Input: $(X_1 : Z_1) = \mathbf{x}(P) \in \mathbb{P}^1$ and $(\hat{A} : \hat{C}) = (a - 2 : 4) \in \mathbb{P}^1$
    Output: $((X_1 : Z_1), \ldots, (X_d : Z_d)) = (\mathbf{x}(P), \mathbf{x}([2]P), \ldots, \mathbf{x}([d]P)) \in (\mathbb{P}^1)^d$
    Cost: $4(d - 1)\mathbf{M} + 2(d - 1)\mathbf{S} + 2(3d - 4)\mathbf{a}$.
1 **if** $d \geq 2$ **then** $(X_2 : Z_2) \leftarrow$ xDBL$((X_1 : Z_1), (\hat{A} : \hat{C}))$
    // 4**M**+2**S**+4**a**
2 **for** $i = 3$ **to** d **do**
3 $(X_i : Z_i) \leftarrow$ xADD$((X_{i-1} : Z_{i-1}), (X_1 : Z_1), (X_{i-2} : Z_{i-2}))$ // 4**M**+2**S**+6**a**
4 **end**
5 **return** $((X_1 : Z_1), \ldots, (X_d : Z_d))$

Algorithm 3. OddIsogeny: $(K^2)^d \times \mathbb{P}^1 \to \mathbb{P}^1$.

    Input: $((\hat{X}_1, \hat{Z}_1), \ldots, (\hat{X}_d, \hat{Z}_d))$ and $(X : Z) \in \mathbb{P}^1$
    Output: $(X' : Z') \in \mathbb{P}^1$ corresponding to $\mathbf{x}(\phi(Q))$ where $\mathbf{x}(Q) = (X : Z)$
    Cost: $4d\mathbf{M} + 2\mathbf{S} + 2(d + 1)\mathbf{a}$.
1 $(\hat{X}, \hat{Z}) \leftarrow (X + Z, X - Z)$ // 2**a**
2 $(X', Z') \leftarrow$ CrissCross$(\hat{X}_1, \hat{Z}_1, \hat{X}, \hat{Z})$ // Algorithm 1
3 **for** $i = 2$ **to** d **do**
4 $(t_0, t_1) \leftarrow$ CrissCross$(\hat{X}_i, \hat{Z}_i, \hat{X}, \hat{Z})$ // Algorithm 1
5 $(X', Z') \leftarrow (t_0 \cdot X', t_1 \cdot Z')$ // 2**M**
6 **end**
7 $(X', Z') \leftarrow (X \cdot (X')^2, Z \cdot (Z')^2)$ // 2**M**+2**S**
8 **return** $(X' : Z')$

---

single isogeny that takes as input the modified set of kernel point coordinates: OddIsogeny is given in Algorithm 3.

    We are now in a position to present SimultaneousOddIsogeny, which is the main algorithm – see Algorithm 4. It takes as input $\mathbf{x}(P) = (X_1 : Z_1) \in \mathbb{P}^1$ and $(\hat{A} : \hat{C}) = (a - 2 : 4)$, which correspond to a point P of order ℓ on $E/K : by^2 = x^3 + ax^2 + x$, as well as an n-tuple $(\mathbf{x}(Q_1), \ldots \mathbf{x}(Q_n)) = ((\mathbf{X}_1 : \mathbf{Z}_1), \ldots, (\mathbf{X}_n : \mathbf{Z}_n)) \in (\mathbb{P}^1)^n$ where the $Q_i \in E$ are such that $Q \notin \langle P \rangle$. The output is an n-tuple corresponding to $(\mathbf{x}(\phi(Q_1)), \ldots \mathbf{x}(\phi(Q_n))) \in (\mathbb{P}^1)^n$, where $\ker(\phi) = \langle P \rangle$.

Simplified Odd-Degree Isogenies in SIDH. Together with an algorithm for computing the multiplication-by-ℓ map, Algorithm 4 is essentially all that is needed to compute an odd ℓ^e-degree isogeny in the context of SIDH. Regardless of which high-level strategy is used to compute the ℓ^e-isogeny (i.e., whether it be the multiplication-based approach [21, Fig. 2] or the *optimal* strategy [15, Sect. 4.2.2]), Algorithm 4 will be called e times to compute e isogenies of degree ℓ. In Algorithm 5 we show how SimultaneousOddIsogeny is to be used in conjunction with the simple conversion function a_from_alpha from Sect. 4 and the Montgomery ladder for computing the multiplication-by-ℓ map. We assume the use

---

Algorithm 4. SimultaneousOddIsogeny: $\mathbb{P}^1 \times \mathbb{P}^1 \times (\mathbb{P}^1)^n \to (\mathbb{P}^1)^n$.

Input: $(X_1 : Z_1) \in \mathbb{P}^1$, $(\hat{A} : \hat{C}) \in \mathbb{P}^1$, and $((\mathbf{X}_1 : \mathbf{Z}_1), \ldots, (\mathbf{X}_n : \mathbf{Z}_n)) \in (\mathbb{P}^1)^n$

Output: $((\mathbf{X}'_1 : \mathbf{Z}'_1), \ldots, (\mathbf{X}'_n : \mathbf{Z}'_n)) \in (\mathbb{P}^1)^n$

Cost: $4(d(n+1) - 1)\mathbf{M} + 2(n + d - 1)\mathbf{S} + 2((d+1)n + (3d-4))\mathbf{a}$.

1 $((X_1 : Z_1), \ldots, (X_d : Z_d)) \leftarrow$ KernelPoints$((X_1 : Z_1), (\hat{A}, \hat{C}))$ // Algorithm 2

2 $((\hat{X}_1, \hat{Z}_1), \ldots, (\hat{X}_d, \hat{Z}_d)) \leftarrow ((X_1 + Z_1, X_1 - Z_1), \ldots, (X_d + Z_d, X_d - Z_d))$ // $2d\mathbf{a}$

3 **for** $j = 1$ **to** n **do**

4     $(\mathbf{X}'_j : \mathbf{Z}'_j) \leftarrow$ OddIsogeny$((\hat{X}_1, \hat{Z}_1), \ldots, (\hat{X}_d, \hat{Z}_d)), (\mathbf{X}_j : \mathbf{Z}_j))$
    // Algorithm 3

5 **end**

6 **return** $((\mathbf{X}'_1 : \mathbf{Z}'_1), \ldots, (\mathbf{X}'_n : \mathbf{Z}'_n))$

---

of the function LADDER as discussed in Sect. 4, where the Montgomery coefficient a is passed in projectively as $(\hat{A} : \hat{C}) = (a - 2 : 4)$; i.e.,

$$\text{LADDER} : \mathbb{P}^1 \times \mathbb{P}^1 \times \mathbb{Z} \to \mathbb{P}^1, \quad (\mathbf{x}(P), (\hat{A} : \hat{C}), \ell^z) \mapsto \mathbf{x}([\ell^z]P). \qquad (18)$$

For ease of exposition, we adopt the multiplication-based approach [21, Fig. 2] for computing the degree ℓ^e-isogeny, but note that the way in which the proposed algorithms are called in Lines 4–6 of Algorithm 5 is analogous if the optimal strategy mentioned above is used; the only difference worth mentioning is that the length of the list of the $(\mathbf{X}'_i : \mathbf{Z}'_i)$ passed in and out of SimultaneousOddIsogeny on Line 6 can change when it is called within the code executing the optimal strategy.

In the notation of Sect. 2, let $E/\mathbb{F}_{p^2} : y^2 = x(x - \alpha)(x - 1/\alpha)$ be the public starting curve in the SIDH protocol. For public key generation, Alice would compute her secret kernel generator as $R_A = [u_A]P_A + [v_A]Q_A$ of order $\ell_A^{e_A}$ (see [15, Algorithm 1]), and with Bob's public basis P_B and Q_B, she can then compute her public key by calling Algorithm 5 as

$$\big((X_{\alpha,A} : Z_{\alpha,A}), ((\mathbf{x}(\phi_A(P_B)), \mathbf{x}(\phi_A(Q_B)), \mathbf{x}(\phi_A(Q_B - P_B)))\big)$$
$$= \text{SIDH\_Isogeny}\big(\mathbf{x}(R_A), (\ell_A, e_A), (\alpha : 1), (\mathbf{x}(P_B), \mathbf{x}(Q_B), \mathbf{x}(Q_B - P_B))\big),$$

where $\ker(\phi_A) = \langle R_A \rangle$, and where $\mathbf{x}(Q_B - P_B)$ is included as an input to avoid sign ambiguities in the subsequent shared secret computations – see [12]. Alice would normalise each of these elements, i.e., convert them all from $\mathbb{P}^1(\mathbb{F}_{p^2})$ into $\mathbb{F}_{p^2}$ via a simultaneous inversion [30], then send them to Bob. Writing $\alpha_A = X_{\alpha,A}/Z_{\alpha,A}$, Bob can then compute $\mathbf{x}(S_B) = \mathbf{x}([u_B]\phi_A(P_B) + [v_B]\phi_A(Q_B))$, and compute the shared secret by calling Algorithm 5 as

$$(X_{\alpha,AB} : Z_{\alpha,AB}) = \text{SIDH\_Isogeny}\big(\mathbf{x}(S_B), (\ell_B, e_B), (\alpha_A : 1)\big),$$

before computing the j-invariant of the Montgomery curve whose coefficient is the output of the function a_from_alpha$((X_{\alpha,AB} : Z_{\alpha,AB}))$. Note that, during the shared secret computation, the $(\mathbb{P}^1)^k$ input to SIDH_Isogeny is empty, i.e., has $k = 0$.

Algorithm 5. SIDH_Isogeny: $\mathbb{P}^1 \times \mathbb{Z}^2 \times \mathbb{P}^1 \times (\mathbb{P}^1)^k \to \mathbb{P}^1 \times (\mathbb{P}^1)^k$.

Input: $\mathbf{x}(P) = (X_1 : Z_1) \in \mathbb{P}^1$, and $(\ell, e) \in \mathbb{Z}^2$, where $|\langle P \rangle| = \ell^e$ on E.
$(\alpha : 1) \in \mathbb{P}^1$, where $\mathrm{ord}((\alpha, 0)) = 2$ on E and $\alpha \neq 0$.
$(\mathbf{x}(Q_1), \dots \mathbf{x}(Q_k)) = ((\mathbf{X}_1 : \mathbf{Z}_1), \dots, (\mathbf{X}_k : \mathbf{Z}_k)) \in (\mathbb{P}^1)^k$, where $Q_i \in$
E and $Q \notin \langle P \rangle$.

Output: $(X_{\alpha'} : Z_{\alpha'}) \in \mathbb{P}^1$, where $\mathrm{ord}((X_{\alpha'}/Z_{\alpha'}, 0)) = 2$ on $\phi(E')$ for $\ker(\phi) =$
$\langle P \rangle$ and $X_{\alpha'} \neq 0$.
$(\mathbf{x}(\phi(Q_1)), \dots \mathbf{x}(\phi(Q_k))) = ((\mathbf{X}_1' : \mathbf{Z}_1'), \dots, (\mathbf{X}_k' : \mathbf{Z}_k')) \in (\mathbb{P}^1)^k$

---

1 $((X_{\alpha'} : Z_{\alpha'}), (\mathbf{X}_1' : \mathbf{Z}_1'), \dots, (\mathbf{X}_k' : \mathbf{Z}_k')) \leftarrow ((\alpha : 1), (\mathbf{X}_1 : \mathbf{Z}_1), \dots, (\mathbf{X}_k : \mathbf{Z}_k))$
   // Initialise
2 $(X_R : Z_R) \leftarrow (X_1 : Z_1)$ // Initialise
3 for $z = e - 1$ **downto** 0 **do**
4 $(\hat{A} : \hat{C}) \leftarrow$ a_from_alpha$((X_{\alpha'} : Z_{\alpha'}))$ // See Sect. 4
5 $(X_S : Z_S) \leftarrow$ LADDER$((X_R : Z_R), (\hat{A} : \hat{C}), \ell^z)$ // See Eq. (18)
6 $((X_R : Z_R), (X_{\alpha'} : Z_{\alpha'}), (\mathbf{X}_1' : \mathbf{Z}_1'), \dots, (\mathbf{X}_k' : \mathbf{Z}_k')) \leftarrow$
    SimultaneousOddIsogeny   // Algorithm 4
    $((X_S : Z_S), (\hat{A} : \hat{C}), ((X_R : Z_R), (X_{\alpha'} : Z_{\alpha'}), (\mathbf{X}_1' : \mathbf{Z}_1'), \dots, (\mathbf{X}_k' : \mathbf{Z}_k')))$
7 end
8 return $(X_{\alpha'} : Z_{\alpha'}), ((\mathbf{X}_1' : \mathbf{Z}_1'), \dots, (\mathbf{X}_n' : \mathbf{Z}_n'))$

We note that the operation counts presented in Algorithms 2–4 do not apply to the special case of $d = 1$. Although Algorithm 4 still performs the 3-isogeny computations in the same number of operations as the dedicated formulas in Appendix A, the claimed operation counts only hold if KernelPoints is called, which is not the case for 3-isogenies (where no additional kernel elements are required).

We conclude this section with a remark on a more compact version of Algorithm 4.

Remark 4 (A low-storage version). The description of the general odd-degree isogeny function in Algorithm 4 aims to minimise the total number of field operations needed for an ℓ-isogeny computation and its evaluation at an arbitrary number of points. However, the recycling of the additions computed in (17) requires us to generate the entire list of d kernel elements before entering the loop that repeatedly calls Algorithm 3. If d is large, the space required to store d elements in $\mathbb{F}_{p^2}$ might become infeasible, especially given the size of the fields used in real-world SIDH implementations. Moreover, this recycling only saves $\mathbb{F}_{p^2}$ additions, and our benchmarking of the SIDH v2.0 software accompanying [12] in the following section revealed that their software has $1\mathbf{M} \approx 20\mathbf{a}$, which means the above recycling will only have a minor benefit on the overall performance. Thus, a more streamlined version of Algorithm 4 would simply compute one of the elements $\mathbf{x}([i]P) = (X_i : Z_i)$ at a time and absorb its contribution to (17) immediately before calling xADD to replace it by $\mathbf{x}([i + 1]P) = (X_{i+1} : Z_{i+1})$, and so on, with no need for Algorithm 2. Since the required storage would then

remain fixed as d increases, this would give a much more compact algorithm for larger d, both in its description and in terms of the storage required to implement it.

6 Implementation Results and Implications

In this section we provide benchmarks for `SimultaneousOddIsogeny`, i.e., the general odd-degree isogeny function in Algorithm 4. We stress that we are not aiming to outperform the relative performance of the 3- and 4-isogenies, by pointing out that the relative performance of odd ℓ-isogenies decreases as ℓ grows larger. The point of this paper is to broaden the class of curves for which SIDH is practical in all of the relevant aspects, i.e., memory requirements, code size, simplicity of the implementation, as well as efficiency. Nevertheless, there are scenarios where larger odd-degree isogenies could be preferred over the low degree ones, as we will discuss later in this section.

Table 1 presents benchmarks for the evaluation of isogenies of degree $\ell \in \{3, 5, \ldots, 15\}$ at $n \in \{1, 2, 5, 8\}$ input points. These timings were obtained by wrapping Algorithm 4 around the SIDH v2.0 software[8] accompanying [11, 12]; this software uses the supersingular isogeny class containing the curve $E/\mathbb{F}_{p^2}: y^2 = x^3 + x$ where $p = 2^{372} \cdot 3^{239} - 1$, where all curves in the class have cardinality $(2^{372} \cdot 3^{239})^2$. We note that this curve does not have $\mathbb{F}_{p^2}$-rational points of order ℓ for odd $\ell > 3$, but this is immaterial; the timings for Algorithm 4 would be exactly the same when working with a curve with rational ℓ-torsion over the same field. We benchmarked in this way in order to get a fair comparison of the cost of different values of ℓ and n when the field arithmetic stays fixed at a size that is relevant to real-world SIDH implementations. We discuss the influence of needing rational ℓ-torsion on the field arithmetic later in this section. The reason we chose to benchmark $n \in \{1, 2, 5, 8\}$ is based on the average number of isogeny evaluations for both Alice and Bob at each step of the SIDH v2.0 software that uses the optimal *tree traversal* (see Sect. 2 or [15, Sect. 4.2.2]) in the main loop: Alice and Bob use roughly 7.15 and 7.70 evaluations of every 4- and 3-isogeny (respectively) during key generation, and this would include one more evaluation if our 2-torsion technique from Sect. 4 was employed (hence $n = 8$), and they use 4.15 and 4.70 respective isogeny evaluations per step during the shared secret phase (hence $n = 5$). We also include $n = 1$ to benchmark the cost of a single isogeny evaluation and $n = 2$ assuming a single isogeny evaluation is included alongside the 2-torsion technique from Sect. 4; this is to view the relative performance of the simple SIDH loop in Algorithm 5 that evaluates each isogeny at one point during the main loop.

Table 1 shows a natural increase in latency as ℓ grows. A single 5-isogeny evaluation costs around 2.71x that of a single 3-isogeny, and the cost of a 15-isogeny evaluation is around 11.40x that of a 3-isogeny. Due to the multiple isogeny evaluations sharing computations performed on the kernel elements (see Sect. 5), naturally these ratios become slightly more favourable for larger ℓ as n

---

[8] See https://github.com/Microsoft/PQCrypto-SIDH.

Table 1. Cycle counts for `SimultaneousOddIsogeny` for different values of ℓ and n. Timing benchmarks were taken on an Intel Core i7-6500U Skylake processor running Ubuntu 14.04.5 LTS with TurboBoost disabled and all cores but one are switched-off. To obtain the executables, we used GNU-`gcc` version 4.8.4 with the `-O2` flag set and GNU assembler version 2.24.

d	ℓ	$n = 1$	$n = 2$	$n = 5$	$n = 8$
1	3	$9,780$	$19,420$	$48,270$	$76,930$
2	5	$26,450$	$43,420$	$93,400$	$143,670$
3	7	$43,310$	$67,490$	$139,280$	$219,270$
4	9	$60,170$	$91,390$	$184,480$	$277,700$
5	11	$77,000$	$115,490$	$230,070$	$344,220$
6	13	$93,710$	$139,370$	$275,170$	$411,800$
7	15	$110,510$	$163,480$	$320,460$	$477,980$

increases: the evaluation of a 5-isogeny (resp. 15-isogeny) at $n = 8$ points costs around 2.03x (resp. 7.18x) the same number of 3-isogeny evaluations. These numbers are depicted graphically on the left of Fig. 1, and the approximate relative slowdown of using ℓ-isogenies within the SIDH framework is depicted on the right. An analogous version of Fig. 1 for ℓ up to $\ell = 301$ is given in Fig. 2. In both figures the cycle counts have been divided by n in order to give a cost per isogeny evaluation.
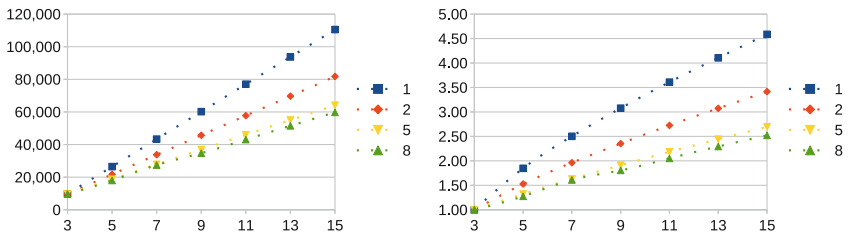

Fig. 1. Average cycle counts for `SimultaneousOddIsogeny` for different values of ℓ and n. Timing benchmarks were taken on an Intel Core i7-6500U Skylake processor running Ubuntu 14.04.5 LTS with TurboBoost disabled and all cores but one are switched-off. To obtain the executables, we used GNU-`gcc` version 4.8.4 with the `-O2` flag set and GNU assembler version 2.24. Raw cycle counts per isogeny evaluation are given on the left, while on the right they are scaled by the factor $\log(3)/(\log(\ell) \cdot \mathbf{C}_3)$, where $\mathbf{C}_3$ is the cost of a 3-isogeny, in order to approximate the relative factor slowdown within the SIDH framework.

The right graphs in Figs. 1 and 2 aim to depict the relative factor slowdowns of computing an ℓ^e isogeny versus a 3^{e_3} isogeny assuming that $\ell^e \approx 3^{e_3}$. However, we must note that a more accurate depiction of the relative slowdown in the
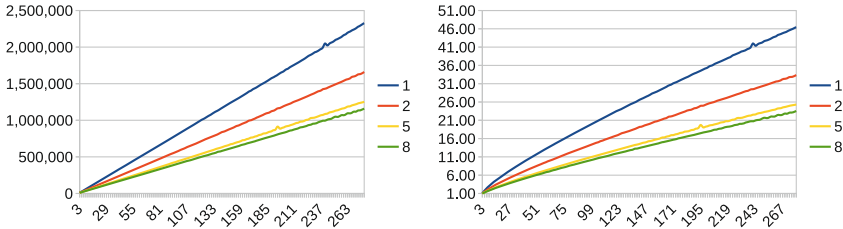
Fig. 2. Average cycle counts for `SimultaneousOddIsogeny` for different values of ℓ and n. Timing benchmarks were taken on an Intel Core i7-6500U Skylake processor running Ubuntu 14.04.5 LTS with TurboBoost disabled and all cores but one are switched-off. To obtain the executables, we used GNU-`gcc` version 4.8.4 with the `-O2` flag set and GNU assembler version 2.24. Raw cycle counts per isogeny evaluation are given on the left, while on the right they are scaled by the factor $\log(3)/(\log(\ell) \cdot \mathbf{C}_3)$, where $\mathbf{C}_3$ is the cost of a 3-isogeny, in order to approximate the relative factor slowdown within the SIDH framework.

SIDH framework would incorporate the relative costs of the multiplication-by-ℓ functions, since these are called almost as frequently as the ℓ-isogeny functions in an optimised implementation (and significantly more times than the ℓ-isogeny functions in the simple SIDH loop – see [12, Sect. 6]). To that end, we point out that the relative slowdown of using ℓ-isogenies would be much less than these graphs depict (as ℓ increases), under the assumption that the Montgomery ladder is called to compute $\mathbf{x}(P) \mapsto \mathbf{x}([\ell]P)$. Table 2 and Fig. 3 exhibit the obvious trend in `LADDER`'s performance as ℓ increases: unlike the linear increase in ℓ-isogeny latencies, the performance of ladder is asymptotically logarithmic, being (roughly) fixed by the value $\lceil \log_2(\ell) \rceil$. In any case, we make the obvious comment that a practically meaningful representation of the performance trade-offs for different values of ℓ can only be obtained by benchmarking similarly optimised implementations in all cases. As we discuss below, such implementations might call for vastly different styles of field arithmetic, so we leave this open for future work.

Implications. At a first glance, Table 1 and Figs. 1 and 2 seem to suggest that, unless faster isogenies of degree $\ell \geq 5$ are found, such higher degree isogenies will not find any meaningful real-world application. However, the ability to compute arbitrary degree isogenies in SIDH already opens up some interesting possibilities as we now discuss.

Firstly, recent work by Bos and Friedberger [8] studied SIDH-friendly primes of the form $p = 2^i r^j - 1$, where r can be any small prime. They investigated a number of different arithmetic techniques, and interestingly, when implementing arithmetic over the field with $p = 2^{372} 3^{329} - 1$ above, found that arithmetic over a comparably sized field $p = 2^{391} 19^{88} - 1$ was actually significantly faster [8, Table 3]. The more severe slowdown of 19-isogenies versus 3-isogenies means that, overall, the performance of 3-isogenies will still be preferred. However,

Table 2. Cycle counts for $[\ell](X : Z)$ on Montgomery Ladder with projective inputs: $(X : Z)$ and $(A_{24} : C_{24})$. Timing benchmarks were taken on an Intel Core i7-6500U Skylake processor running Ubuntu 14.04.5 LTS with TurboBoost disabled and all cores but one are switched-off. To obtain the executables, we used GNU-`gcc` version 4.8.4 with the `-O2` flag set and GNU assembler version 2.24. For the fixed odd low degrees of $\ell \in \{3, 5, 7\}$, we also present the cycle counts of our own optimised, dedicated algorithms for computing the multiplication-by-ℓ maps, since this might be of interest for future implementers; see Appendix A for justification.

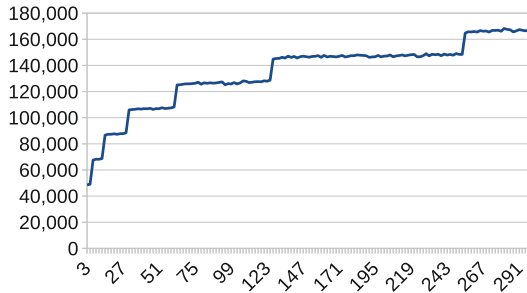| Operation	ladder	Optimized
$[2](X : Z)$	-	9,608
$[3](X : Z)$	28,954	18,622
$[5](X : Z)$	48,603	27,346
$[7](X : Z)$	49,086	36,110
$[9](X : Z)$	67,610	-
$[11](X : Z)$	68,429	-
$[13](X : Z)$	68,125	-
$[15](X : Z)$	68,848	-
$[17](X : Z)$	86,717	-

Fig. 3. Cycle counts chart for $[\ell](X : Z)$ using Montgomery ladder with projective inputs: $(X : Z)$ and $(A_{24} : C_{24})$. Timing benchmarks were taken on an Intel Core i7-6500U Skylake processor running Ubuntu 14.04.5 LTS with TurboBoost disabled and all cores but one are switched-off. To obtain the executables, we used GNU-`gcc` version 4.8.4 with the `-O2` flag set and GNU assembler version 2.24.

in real-world applications like the transport layer security (TLS) protocol, it is typically one side of the protocol (i.e., the server, who is processing many SIDH instances) where performance is the bottleneck, while the performance of a single SIDH instance on the client side is ultimately a non-issue. In such a situation, we could envision affording the server the luxury of the faster prime $p = 2^{391}19^{88} - 1$ *and* the faster 4-isogenies in order to get the best of both worlds, while the client could put up with the 19-isogenies and not be noticibly hampered by the increased latency on their side.

Another possibility opened up by Algorithm 4 is the abandonment of even-degree isogenies on either side of the protocol, in the name of implementation simplicity. For example, SIDH implementations using primes of the form[9] $p = 4 \cdot 3^i 5^j - 1$ could be implemented using Algorithm 4 for isogenous curve and point operations on both sides. This would make for a much simpler and more compact code-base, and could be an attractive option if the relatively modest slowdown from 4- to 5-isogenies (and the possible slowdown of the new shaped primes) is justifiable.

Finally, we leave it as an open question to see whether primes *not* of the form $p = f \cdot 2^i 3^j - 1$ can be found where arithmetic is fast enough to justify isogenies of $\ell \geq 5$. It could even be possible to find fast primes where $p \pm 1$ is smooth but contains many small, unique prime factors, and where isogeny walks on either or both sides of the protocol involve isogenies of different degrees. Of course, the security implications of such a choice are also left as open.

A   Improvements for 3- and 4-isogenies

In this section, we briefly present improved explicit formulas, operation counts, and performance benchmarks for 3- and 4-isogeny and related operations in SIDH; this provides implementers with a fair comparison of the general ℓ-isogeny algorithm and more optimised formulas for the currently used 3- and 4-isogenies. A complete list of the improved operations are presented in Table 3 and the cycle counts are compared in Table 4; all of the associated explicit formulas are presented thereafter. We plugged these formulas into the SIDH v2.0 library from [12] and Table 5 gives the overall improvements of each stage of the SIDH key exchange protocol.

Below we use $(X_3 : Z_3)$ and $(X_4 : Z_4)$ to denote the coordinates of points P_3 and P_4 of orders 3 and 4 respectively, under the **x** map. We use $(A_{24} : C_{24})$ to denote the projective version of the Montgomery curve constant $(a - 2)/4$, i.e., $(A_{24} : C_{24}) = (a - 2 : 4)$ in $\mathbb{P}^1$.

The `3_iso_curve` operation

$$(A'_{24} : C'_{24}) = \big((X_3 + Z_3)(Z_3 - 3X_3)^3 : 16X_3 Z_3^3\big)$$

takes $2\mathbf{M} + 3\mathbf{S} + 14\mathbf{a}$ and produces the common subexpressions $K_1 = X_3 - Z_3$ and $K_2 = X_3 + Z_3$. The justification of the claimed operation count is as follows:

$K_1 = X_3 - Z_3,\ R_1 = K_1^2,\ K_2 = X_3 + Z_3,\ R_2 = K_2^2,\ R_3 = R_2 + R_1,$
$R_4 = K_1 + K_2,\ R_4 = R_4^2,\ R_4 = R_4 - R_3,\ R_3 = R_4 + R_2,\ R_4 = R_4 + R_1,$
$R_5 = R_1 + R_4,\ R_5 = 2R_5,\ R_5 = R_5 + R_2,\ A'_{24} = R_5 \cdot R_3,\ R_5 = R_2 + R_3,$
$R_5 = 2R_5,\ R_5 = R_5 + R_1,\ R_5 = R_5 \cdot R_4,\ C'_{24} = R_5 - A'_{24}.$

---

[9] We still need the cofactor of 4 in the group order to be able to exploit the Montgomery form – see Sect. 2.

Table 3. Operation counts comparison for common elliptic curve and isogeny functions within existing SIDH implementations.

	xTPL	3_iso_point	3_iso_curve	4_iso_point	4_iso_curve
CLN2016	8M+4S+8a	6M+2S+2a	3M+3S+8a	9M+1S+6a	5S+7a
This work	9M+2S+17a	4M+2S+4a	2M+3S+14a	6M+2S+6a	4S+4a
	7M+5S+11a				

Table 4. Cycle counts comparison for common elliptic curve and isogeny functions within existing SIDH implementations. Timing benchmarks were taken on an Intel Core i7-6500U Skylake processor running Ubuntu 14.04.5 LTS with TurboBoost disabled and all cores but one are switched-off. To obtain the executables, we used GNU-`gcc` version 4.8.4 with the `-O2` flag set and GNU assembler version 2.24.

Operation	CLN2016	This work	Speed-up
xTPL	19,226	18,622	1.032x
3_iso_curve	9,264	8,202	1.129x
3_iso_point	12,901	9,581	1.346x
4_iso_curve	6,276	5,095	1.232x
4_iso_point	17,432	13,545	1.287x

Table 5. Cycle counts for Ephemeral isogeny-based key exchange. Timing benchmarks were taken on an Intel Core i7-6500U Skylake processor running Ubuntu 14.04.5 LTS with TurboBoost disabled and all cores but one are switched-off. To obtain the executables, we used GNU-`gcc` version 4.8.4 with the `-O2` flag set and GNU assembler version 2.24.

Operation	CLN2016	This work	Speed-up
Alice's key generation	39,043,000	33,266,000	1.174x
Bob's key generation	44,289,000	37,430,000	1.183x
Alice's shared key computation	36,716,000	33,240,000	1.105x
Bob's shared key computation	42,576,000	38,046,000	1.120x

The `3_iso_point` operation

$$(X' : Z') = \left(X(X_3 X - Z_3 Z)^2 : Z(Z_3 X - X_3 Z)^2 \right)$$

takes $4\mathbf{M} + 2\mathbf{S} + 4\mathbf{a}$ and uses the common subexpressions $K_1 = X_3 - Z_3$ and $K_2 = X_3 + Z_3$. The justification of the claimed operation count is as follows:

$R_2 := X + Z$, $R_3 := R_2 \cdot K_1$, $R_2 := X - Z$, $R_1 := R_2 \cdot K_2$, $R_2 := R_1 + R_3$,
$R_2 := R_2^2$, $X' := R_2 \cdot X$, $R_1 := R_3 - R_1$, $R_1 := R_1^2$, $Z' := R_1 \cdot Z$.

The `4_iso_curve` operation

$$(A'_{24} : C'_{24}) = \left(X_4^4 - Z_4^4 : Z_4^4 \right)$$

takes $4\mathbf{S} + 4\mathbf{a}$ and produces the common subexpressions $K_1 = 4Z_4^2$, $K_2 = X_4 - Z_4$, and $K_3 = X_4 + Z_4$. The justification of the claimed operation count is as follows:

$$K_1 = Z_4^2,\ R_1 = X_4^2,\ R_1 = R_1^2,\ C_{24}' = K_1^2,\ A_{24}' = R_1 - C_{24}',\ K_1 = 4K_1,$$
$$K_2 = X_4 - Z_4,\ K_3 = X_4 + Z_4.$$

The `4_iso_point` operation

$$(X' : Z') = \big(X \left(2X_4 Z_4 Z - (X_4^2 + Z_4^2) X \right) (X_4 X - Z_4 Z)^2 :$$
$$Z \left(2X_4 Z_4 X - (X_4^2 + Z_4^2) Z \right) (Z_4 X - X_4 Z)^2 \big)$$

takes $6\mathbf{M} + 2\mathbf{S} + 6\mathbf{a}$ and benefits from the common subexpressions $K_1 = 4Z_4^2$, $K_2 = X_4 - Z_4$, and $K_3 = X_4 + Z_4$ generated by `4_iso_curve`. The justification of the claimed operation count is as follows:

$$R_2 = X + Z,\ R_3 = R_2 \cdot K_2,\ R_4 = X - Z,\ R_1 = R_4 \cdot K_3,\ R_2 = R_4 \cdot R_2,$$
$$R_4 = R_1 + R_3,\ R_4 = R_4^2,\ R_3 = R_1 - R_3,\ R_3 = R_3^2,\ R_2 = K_1 \cdot R_2,\ R_1 = R_4 + R_2,$$
$$R_2 = R_3 - R_2,\ X' = R_4 \cdot R_1,\ Z' = R_3 \cdot R_2.$$

The `xTPL` operation

$$[3](X : Z) = \big(X(16 A_{24} X Z^3 - C_{24}(X - 3Z)(X + Z)^3)^2 :$$
$$Z(16 A_{24} X^3 Z + C_{24}(3X - Z)(X + Z)^3)^2 \big)$$

takes $9\mathbf{M} + 2\mathbf{S} + 16\mathbf{a}$ assuming that $K_1 = A_{24} + C_{24}$ is cached. The justification of the claimed operation count is as follows:

$$R_1 = X - Z,\ R_2 = R_1^2,\ R_3 = X + Z,\ R_4 = R_3^2,\ R_5 = R_4 + R_2,\ R_6 = R_2 - R_4,$$
$$R_7 = R_4 \cdot K_1,\ R_8 = R_2 \cdot A_{24},\ R_4 = R_8 + R_7,\ R_2 = R_7 - R_8,\ R_4 = R_4 \cdot R_6,$$
$$R_5 = R_2 \cdot R_5,\ R_2 = R_2 \cdot R_6,\ R_2 = 2R_2,\ R_6 = R_4 + R_5,\ R_5 = R_4 - R_5,$$
$$R_4 = R_6 + R_2,\ R_6 = R_6 - R_2,\ R_4 = R_4 \cdot R_6,\ R_6 = R_2 \cdot R_5,\ R_6 = 2R_6,$$
$$R_5 = R_4 - R_6,\ R_4 = R_4 + R_6,\ R_2 = R_4 \cdot R_3,\ R_1 = R_1 \cdot R_5,\ X_{out} = R_2 + R_1,$$
$$Z_{out} = R_2 - R_1.$$

Alternatively, the `xTPL` operation takes $7\mathbf{M} + 5\mathbf{S} + 10\mathbf{a}$ assuming that $K_1 = A_{24} + C_{24}$ is cached. The justification of the claimed operation count is as follows:

$$R_1 = X - Z,\ R_3 = R_1^2,\ R_2 = X + Z,\ R_4 = R_2^2,\ R_5 = R_2 + R_1,\ R_1 = R_2 - R_1,$$
$$R_2 = R_5^2,\ R_2 = R_2 - R_4,\ R_2 = R_2 - R_3,\ R_6 = R_4 \cdot K_1,\ R_4 = R_6 \cdot R_4,$$
$$R_7 = R_3 \cdot A_{24},\ R_3 = R_3 \cdot R_7,\ R_4 = R_3 - R_4,\ R_3 = R_6 - R_7,\ R_2 = R_3 \cdot R_2,$$
$$R_3 = R_4 + R_2,\ R_3 = R_3^2,\ X_{out} = R_3 \cdot R_5,\ R_2 = R_4 - R_2,\ R_2 = R_2^2,$$
$$Z_{out} = R_2 \cdot R_1.$$

The dedicated explicit formulas for the multiplication-by-5 map referred to in Table 2 take $(X_{out} : Z_{out}) = \mathbf{x}([5]P)$, where $\mathbf{x}(P) = (X : Z)$, and are as below. The cost is $11\mathbf{M} + 6\mathbf{S} + 14\mathbf{a}$, assuming that $K_1 = A_{24} + C_{24}$ is cached.

$$R_1 = X - Z,\ R_2 = R_1^2,\ R_3 = X + Z,\ R_4 = R_3^2,\ R_5 = R_4 + R_2,\ R_1 = R_1 + R_3,$$
$$R_1 = R_1^2,\ R_3 = R_1 - R_5,\ R_2 = R_2 \cdot A_{24},\ R_4 = K_1 \cdot R_4,\ R_4 = R_4 - R_2,$$
$$R_5 = R_4 \cdot R_5,\ R_2 = R_2^2,\ R_1 = R_2 \cdot C_{24},\ R_2 = R_1/4,\ R_2 = R_5 - R_2,$$
$$R_5 = R_4 \cdot R_3,\ R_3 = R_2 + R_5,\ R_4 = R_2 - R_5,\ R_3 = R_3 \cdot R_4,\ R_4 = R_2 \cdot R_3,$$
$$R_1 = R_1 \cdot R_2,\ R_1 = R_1 + R_3,\ R_3 = R_1 \cdot R_5,\ R_2 = R_4 + R_3,\ R_2 = R_2^2,$$
$$X_{out} = R_2 \cdot X_1,\ R_1 = R_4 - R_3,\ R_1 = R_1^2,\ Z_{out} = R_1 \cdot Z_1.$$

The dedicated explicit formulas for the multiplication-by-7 map referred to in Table 2 take $(X_{out} : Z_{out}) = \mathbf{x}([7]P)$, where $\mathbf{x}(P) = (X : Z)$, and are as below. The cost is $14\mathbf{M} + 9\mathbf{S} + 18\mathbf{a}$, assuming that $K_1 = A_{24} + C_{24}$ is cached.

$R_1 = X - Z,\ R_3 = R_1^2,\ R_2 = X + Z,\ R_4 = R_2^2,\ R_1 = R_1 + R_2,\ R_1 = R_1^2,$
$R_1 = R_1 - R_4,\ R_2 = R_1 - R_3,\ R_1 = K_1 \cdot R_4,\ R_4 = R_1 \cdot R_4,\ R_5 = R_3 \cdot A_{24},$
$R_1 = R_1 - R_5,\ R_3 = R_3 \cdot R_5,\ R_3 = R_4 - R_3,\ R_4 = R_2 \cdot R_1,\ R_1 = R_3^2,\ R_5 = R_4^2,$
$R_4 = R_3 + R_4,\ R_4 = R_4^2,\ R_2 = R_2^2,\ R_2 = R_2 \cdot R_3,\ R_3 = R_1 - R_5,\ R_1 = R_1 + R_5,$
$R_4 = R_1 - R_4,\ R_1 = R_1 \cdot R_3,\ R_3 = 2R_3,\ R_2 = R_2 \cdot C_{24},\ R_5 = R_2 \cdot R_5,$
$R_5 = R_5 + R_1,\ R_1 = R_3 + R_2,\ R_3 = R_3 \cdot R_5,\ R_2 = R_1 + R_2,\ R_2 = R_2 \cdot R_1,$
$R_1 = 2R_3,\ R_2 = R_4 \cdot R_2,\ R_3 = R_1 + R_2,\ R_3 = R_3^2,\ X_{out} = X_1 \cdot R_3,$
$R_3 = R_1 - R_2,\ R_3 = R_3^2,\ Z_{out} = Z_1 \cdot R_3.$

References

1. Alkim, E., Ducas, L., Pöppelmann, T., Schwabe, P.: Post-quantum key exchange - A new hope. In: Holz, T., Savage, S. (eds.) 25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, 10–12 August 2016, pp. 327–343. USENIX Association (2016)

2. Azarderakhsh, R., Jao, D., Kalach, K., Koziel, B., Leonardi, C.: Key compression for isogeny-based cryptosystems. In: Emura, K., Hanaoka, G., Zhang, R. (eds.) Proceedings of the 3rd ACM International Workshop on ASIA Public-Key Cryptography, AsiaPKC@AsiaCCS, Xi'an, China, 30 May–03 June 2016, pp. 1–10. ACM (2016)

3. Bernstein, D.J., Birkner, P., Joye, M., Lange, T., Peters, C.: Twisted edwards curves. In: Vaudenay, S. (ed.) AFRICACRYPT 2008. LNCS, vol. 5023, pp. 389–405. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-68164-9_26

4. Bernstein, D.J., Buchmann, J., Dahmen, E.: Post-Quantum Cryptography. Springer Science & Business Media, Heidelberg (2009). https://doi.org/10.1007/978-3-540-88702-7

5. Bernstein, D.J., Chou, T., Schwabe, P.: McBits: Fast constant-time code-based cryptography. In: Bertoni, G., Coron, J.-S. (eds.) CHES 2013. LNCS, vol. 8086, pp. 250–272. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40349-1_15

6. Bos, J.W., Costello, C., Ducas, L., Mironov, I., Naehrig, M., Nikolaenko, V., Raghunathan, A., Stebila, D.: Frodo: Take off the ring! Practical, quantum-secure key exchange from LWE. In: Weippl, E.R., Katzenbeisser, S., Kruegel, C., Myers, A.C., Halevi, S. (eds.) Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, 24–28 October 2016, pp. 1006–1018. ACM (2016)

7. Bos, J.W., Costello, C., Naehrig, M., Stebila, D.: Post-quantum key exchange for the TLS protocol from the ring learning with errors problem. In: 2015 IEEE Symposium on Security and Privacy, SP 2015, San Jose, CA, USA, 17–21 May 2015, pp. 553–570. IEEE Computer Society (2015)

8. Bos, J.W., Friedberger, S.: Fast arithmetic modulo $2^x\ p^y \pm 1$. In: Burgess, N., Bruguera, J.D., de Dinechin, F. (eds.) 24th IEEE Symposium on Computer Arithmetic, ARITH 2017, London, United Kingdom, 24–26 July 2017, pp. 148–155. IEEE Computer Society (2017)

9. Castryck, W., Galbraith, S., Farashahi, R.R.: Efficient arithmetic on elliptic curves using a mixed Edwards-Montgomery representation. Cryptology ePrint Archive, Report 2008/218 (2008). http://eprint.iacr.org/2008/218

10. Chou, T.: QcBits: Constant-time small-key code-based cryptography. In: Gierlichs, B., Poschmann, A.Y. (eds.) CHES 2016. LNCS, vol. 9813, pp. 280–300. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53140-2_14

11. Costello, C., Jao, D., Longa, P., Naehrig, M., Renes, J., Urbanik, D.: Efficient compression of SIDH public keys. In: Coron, J.-S., Nielsen, J.B. (eds.) EUROCRYPT 2017. LNCS, vol. 10210, pp. 679–706. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-56620-7_24

12. Costello, C., Longa, P., Naehrig, M.: Efficient algorithms for supersingular isogeny diffie-hellman. In: Robshaw, M., Katz, J. (eds.) CRYPTO 2016. LNCS, vol. 9814, pp. 572–601. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53018-4_21

13. Couveignes, J.: Hard homogeneous spaces. Cryptology ePrint Archive, Report 2006/291 (2006). http://eprint.iacr.org/2006/291

14. Diffie, W., Hellman, M.E.: New directions in cryptography. IEEE Trans. Inf. Theor. **22**(6), 644–654 (1976)

15. De Feo, L., Jao, D., Plût, J.: Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies. J. Math. Cryptol. **8**(3), 209–247 (2014)

16. Fluhrer, S.: Cryptanalysis of ring-LWE based key exchange with key share reuse. Cryptology ePrint Archive, Report 2016/085 (2016). http://eprint.iacr.org/2016/085

17. Fujisaki, E., Okamoto, T.: How to enhance the security of public-key encryption at minimum cost. In: Imai, H., Zheng, Y. (eds.) PKC 1999. LNCS, vol. 1560, pp. 53–68. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-49162-7_5

18. Galbraith, S.D.: Mathematics of Public Key Cryptography. Cambridge University Press, New York (2012)

19. Galbraith, S.D., Petit, C., Shani, B., Ti, Y.B.: On the security of supersingular isogeny cryptosystems. In: Cheon, J.H., Takagi, T. (eds.) ASIACRYPT 2016. LNCS, vol. 10031, pp. 63–91. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53887-6_3

20. Hoffstein, J., Pipher, J., Silverman, J.H.: NTRU: A ring-based public key cryptosystem. In: Buhler, J.P. (ed.) ANTS 1998. LNCS, vol. 1423, pp. 267–288. Springer, Heidelberg (1998). https://doi.org/10.1007/BFb0054868

21. Jao, D., Feo, L.: Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies. In: Yang, B.-Y. (ed.) PQCrypto 2011. LNCS, vol. 7071, pp. 19–34. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-25405-5_2

22. Kirkwood, D., Lackey, B.C., McVey, J., Motley, M., Solinas, J.A., Tuller, D.: Failure is not an option: Standardization issues for post-quantum key agreement. Talk at NIST workshop on Cybersecurity in a Post-Quantum World, April 2015. http://www.nist.gov/itl/csd/ct/post-quantum-crypto-workshop-2015.cfm

23. Kobara, K., Imai, H.: Semantically secure McEliece public-key cryptosystems - conversions for McEliece PKC -. In: Kim, K. (ed.) PKC 2001. LNCS, vol. 1992, pp. 19–35. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-44586-2_2

24. Kocher, P.C.: Timing attacks on implementations of diffie-hellman, RSA, DSS, and other systems. In: Koblitz, N. (ed.) CRYPTO 1996. LNCS, vol. 1109, pp. 104–113. Springer, Heidelberg (1996). https://doi.org/10.1007/3-540-68697-5_9

25. Kohel, D.R.: Endomorphism rings of elliptic curves over finite fields. PhD thesis, University of California, Berkeley (1996)

26. Koziel, B., Azarderakhsh, R., Kermani, M.M., Jao, D.: Post-quantum cryptography on FPGA based on isogenies on elliptic curves. IEEE Trans. Circuits Syst. **64**(1), 86–99 (2017)
27. Lyubashevsky, V., Peikert, C., Regev, O.: On ideal lattices and learning with errors over rings. J. ACM **60**(6), 43:1–43:35 (2013)
28. McEliece, R.J.: A public-key cryptosystem based on algebraic coding theory. Coding Thv **4244**, 114–116 (1978)
29. Misoczki, R., Tillich, J., Sendrier, N., Barreto, P.S.L.M.: MDPC-McEliece: New McEliece variants from moderate density parity-check codes. In: Proceedings of the 2013 IEEE International Symposium on Information Theory, Istanbul, Turkey, 7–12 July 2013, pp. 2069–2073. IEEE (2013)
30. Montgomery, P.L.: Speeding the Pollard and elliptic curve methods of factorization. Math. Comput. **48**(177), 243–264 (1987)
31. Moody, D., Shumow, D.: Analogues of Vélu's formulas for isogenies on alternate models of elliptic curves. Math. Comput. **85**(300), 1929–1951 (2016)
32. Mosca, M.: Cybersecurity in an ERA with quantum computers: will we be ready? Cryptology ePrint Archive, Report 2015/1075 (2015). http://eprint.iacr.org/2015/1075
33. Peikert, C.: Lattice cryptography for the internet. In: Mosca, M. (ed.) PQCrypto 2014. LNCS, vol. 8772, pp. 197–219. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-11659-4_12
34. Regev, O.: On lattices, learning with errors, random linear codes, and cryptography. In: Gabow, H.N., Fagin, R. (ed.) Proceedings of the 37th Annual ACM Symposium on Theory of Computing, Baltimore, MD, USA, 22–24 May 2005, pp. 84–93. ACM (2005)
35. Rostovtsev, A., Stolbunov, A.: Public-key cryptosystem based on isogenies. Cryptology ePrint Archive, Report 2006/145 (2006). http://eprint.iacr.org/
36. Shor, P.W.: Algorithms for quantum computation: Discrete logarithms and factoring. In: 1994 Proceedings and 35th Annual Symposium on Foundations of Computer Science, pp. 124–134. IEEE (1994)
37. Silverman, J.H.: The Arithmetic of Elliptic Curves. Graduate Texts in Mathematics, 2nd edn. Springer, New York (2009). https://doi.org/10.1007/978-1-4757-1920-8
38. Stebila, D., Mosca, M.: Post-quantum key exchange for the Internet and the open quantum safe project. Cryptology ePrint Archive, Report 2016/1017 (2016). http://eprint.iacr.org/2016/1017
39. Stolbunov, A.: Cryptographic Schemes Based on Isogenies. PhD thesis, Norwegian University of Science and Technology (2012)
40. The National Institute of Standards and Technology (NIST). Submission requirements and evaluation criteria for the post-quantum cryptography standardization process, December 2016
41. Vélu, J.: Isogénies entre courbes elliptiques. CR Acad. Sci. Paris Sér. AB **273**, A238–A241 (1971)
42. Washington, L.C.: Elliptic Curves: Number Theory and Cryptography. CRC Press, Boca Raton (2008)
43. Yoo, Y., Azarderakhsh, R., Jalali, A., Jao, D., Soukharev, V.: A post-quantum digital signature scheme based on supersingular isogenies (2017). http://eprint.iacr.org/2017/186. To appear in Financial Cryptography and Data Security