A simple and compact algorithm for SIDH with arbitrary degree isogenies

Craig Costello[1] and Huseyin Hisil[2]

[1] Microsoft Research, 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 compute both 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 auxiliary 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 [39].

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 [37].

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 [29,10].

(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 [35,38], 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 [15,12,2,26,11].

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[3]. 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 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 hardcore 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.

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

– *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 the 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[4], 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 isogeny-based cryptographic schemes, e.g., to the other schemes proposed by De Feo, Jao and Plût [15], and to the recent isogeny-based signature scheme from Yoo *et al.* [41].

Organisation. We give the preliminaries in Section 2. We provide the new formula for odd-degree Montgomery isogenies in Section 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 Section 4, before presenting the general-purpose odd-degree isogeny algorithm in Section 5. We provide implementation benchmarks and conclude with some potential implications in Section 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, Jao and Plût [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, that 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 4 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]

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

Herein 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, 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.

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

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 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]

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

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 can be 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

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

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 $p \mp 1 = (f \cdot n_A n_B)^2$. 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 that 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 P_A and Q_A form a basis for 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 [15,12,2,26,11], $\ell_A = 2$ and $\ell_B = 3$, and Alice computes her 2^{e_A}-isogeny as a composition of 2- and/or 4-isogenies (see [15,12]), 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, Figure 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 [40] 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, Jao and Plût [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 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, §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 Equation (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 how it was derived and of the related work of Moody and Shumow [31].

Theorem 1. *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 $\sigma = \sum_{i=1}^{d}(1/x_{[i]P} - x_{[i]P})$ and $\pi = \prod_{i=1}^{d} x_{[i]P}$. The Montgomery curve*

$$E'/K : B'y^2 = x^3 + A'x^2 + x$$

with

$$A' = (6\sigma + A) \cdot \pi^2 \qquad \text{and} \qquad B' = B \cdot \pi^2 \tag{5}$$

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

$$\phi : (x, y) \mapsto (\phi_x(x), y \cdot \phi_x'(x)),$$

where

$$\phi_x(x) = x \cdot \prod_{i=1}^{d} \left(\frac{x \cdot x_{[i]P} - 1}{x - x_{[i]P}} \right)^2 . \tag{6}$$

Proof. We show that (i) ϕ is a rational map from E to E'; (ii) ϕ is regular at all points and is therefore a morphism; (iii) $\phi(\mathcal{O}_E) = \mathcal{O}_{E'}$ so that ϕ is an isogeny; (iv) $\ker(\phi) = \langle P \rangle$; (v) is normalised.

For (i), we follow [31] and use (6) to derive A' and B' by first observing that ϕ maps the rational point $(0,0) \in E$ to $(0,0) \in E'$ and then writing $G(x,y) = B'(y \cdot \phi'_x(x))^2 - (\phi_x(x)^3 + A'\phi'_x(x)^2 + \phi'_x(x))$. The non-singularity of the point $(0,0)$ on E' implies that $G(x,y)$ has a simple zero at $(0,0)$. Now, write $u(x) = \prod_{i=1}^{d}(x \cdot x_{[i]P} - 1)$ and $v(x) = \prod_{i=1}^{d}(x - x_{[i]P})$ so that $\phi_x(x) = x \cdot u(x)^2/v(x)^2$. It follows that

$$\begin{aligned} v(x)^6 \cdot G(x,y) = (B'/B)(x^3 + Ax^2 + x) \cdot \left(v(x)^3 \cdot \phi'_x(x)\right)^2 \\ - \left((x \cdot u(x)^2)^3 + A'(x \cdot u(x)^2)^2 v(x)^2 + x \cdot u(x)^2 v(x)^4\right), \end{aligned}$$

where

$$v(x)^3 \cdot \phi'_x(x) = u(x)^2 \cdot v(x) + 2x \cdot u(x) \cdot [v(x)u'(x) - u(x)v'(x)]$$

with

$$u'(x) = \sum_{i=1}^{d} \prod_{j \neq i} x_i(xx_j - 1) \qquad \text{and} \qquad v'(x) = \sum_{i=1}^{d} \prod_{j \neq i} (x - x_j). \tag{7}$$

Since $v(0) = (-1)^d \pi$, which cannot be 0, it follows that the right hand side of (7) (which has no constant term) has a simple zero when $x = 0$, and thus that $G(x,y)$ must be identically zero. Writing $v(x)^6 G(x,y) = c_1 x + c_2 x^2 + O(x^3)$ yields $c_1 = (B'\pi^2 - B\pi^4)/B$ and thus $B' = B \cdot \pi^2$. Making this substitution yields $c_2 = \pi^2(A\pi^2 - A' - 6\sigma\pi^2)$ and hence $A' = (6\sigma + A) \cdot \pi^2$ as claimed. By construction, we have now established that ϕ is a rational map from E to E'.

For (ii), ϕ is clearly regular at all points not in $\langle P \rangle$. The points $Q = (x_Q, y_Q)$ in $\langle P \rangle$ of order ℓ, which all have $y_Q \neq 0$, are mapped to the projective point in $\mathbb{P}^2$ equivalent to $(v(x_Q)^3 \cdot \phi_x(x_Q) : v(x_Q)^3 y_Q \cdot \phi'_x(x_Q) : v(x_Q)^3) \sim (0:1:0)$, since $v(x_Q) = 0$ but from (7) we have $v(x_Q)^3 \cdot \phi'(x_Q) = -2u(x_Q)^2 v'(x_Q) \neq 0$. To show that ϕ is regular at $\mathcal{O}_E$, we project E and E' into $\mathbb{P}(1,2,1)$ to represent points as $(X : Y : Z)$, where $x = X/Z$ and $y = Y/Z^2$. Here $\mathcal{O}_E = (1:0:0)$, and on substitution into (6) and (7) we see that $\phi(\mathcal{O}_E) = O_{E'} = (1:0:0)$, which proves (iii). Furthermore, for (iv), the $2d$ points $Q = (x_Q, y_Q)$ of order ℓ in $\langle P \rangle$ give $v(x_Q) = 0$, which shows that $\langle P \rangle \in \ker(\phi)$; conversely, if $Q \in \ker(\phi)$, then $x_Q = x_{[i]P}$ for some $i \in \{1, \ldots, d\}$, so $Q \in \{[i]P, [\ell - i]P\}$ and thus $Q \in \langle P \rangle$, which gives $\ker(\phi) = \langle P \rangle$.

Finally, for (v), ϕ is normalised if the pullback of the invariant differential on E' is equal to the invariant differential on E. The invariant differential on E is $\omega_E = dx/2y$, while from (6) the invariant differential on E' is $\omega_{E'} = d\phi_x(x)/2y\phi'_x(x)$. The result is immediate since $d\phi_x(x) = xd\phi'_x(x)$. □

Remark 2. Write $\langle P \rangle^* = \langle P \rangle \setminus \{\mathcal{O}_E\}$. An alternative way to state the coordinate maps in (6) is

$$\phi(Q) = \left(x_Q \cdot \prod_{T \in \langle P \rangle^*} x_{Q+T} \quad , \quad y_Q \cdot \prod_{T \in \langle P \rangle^*} \frac{x_T \cdot y_{Q+T}}{y_T} \right), \tag{8}$$

where the connection between (6) and (8) can be drawn via (1) and (2) and the symmetries $x_{[i]P} = x_{[\ell-i]P}$ and $y_{[i]P} = -y_{[\ell-i]P}$ inside the order-ℓ subgroup generated by P.

The simplicity of Equation (6), in comparison to Vélu's formulas [40] on general Weierstrass curves, lies in the fact that it factors neatly across the different multiples of P. This lends itself to the simple algorithm we describe in Section 5.

We now turn to describing how the coordinate maps in (6) were derived. It should be first remarked that, given a point of odd order, Vélu's formulas compute coordinate maps that take points on the general Weierstrass curve

$$W: \quad y^2 + a_1 xy + a_3 y = x^3 + a_2 x^2 + a_4 x + a_6$$

to the isogenous curve

$$W': \quad y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4'x + a_6'.$$

Observe that only the coefficients a_4 and a_6 change under the isogeny, which is why the formulas preserve short Weierstrass form in the case where $a_1 = a_3 = a_2 = 0$. In our case, however, Vélu's formulas do not preserve the Montgomery model: we start with $a_1 = a_3 = a_6 = 0$, $a_2 = a$ and $a_4 = 1$, and while Vélu's formulas preserve $a_2 = a$, they output $a_6' \neq 0$ and $a_4' \neq 1$. In general, the generic transformation of such a curve into Montgomery form requires the computation of a square root, which is why De Feo, Jao and Plût [15, Eq. (16)] describe a different transformation that exploits the special 2-torsion point common to all Montgomery curves, as it moves through odd-degree isogenies where its order is preserved[5]. They used this transformation to derive the Montgomery 3-isogeny [15, Eq. (17)], which is the special case of $ell = 3$ in Theorem 1.

We followed the same recipe as De Feo, Jao and Plût [15] in the case of $\ell = 5$ and $\ell = 7$ to see the pattern in (6). We applied Vélu's formulas to a starting Montgomery curve to produce a Weierstrass curve, then applied the transformation in [15, Eq. (16)] to transform the isogenous curve into Montgomery form. At this point the coordinate maps still appeared to be cumbersome, and it took us a number of attempted computer algebra simplifications to arrive at (6).

In particular, applying the above recipe will, in general, mean that the coordinate maps still involve the Montgomery curve constant a. In the case of 3-isogenies, eliminating the curve constant is easy, since the 3-division polynomial $\psi_3(x) = 3x^4 + 4ax^3 + 6x^2 - 1$ can be used to replace a with the x-coordinate of the 3-torsion point. However, in the case of $\ell \geq 5$, the ℓ-division polynomials cannot be used to eliminate a in the same way; observe that higher powers of a appear in (4).

To eliminate a when $\ell = 5$ and $\ell = 7$, we made use of circular relations involving the x-only doubling map

$$[2]_x: \quad x_P \mapsto x_{[2]P} = \frac{(x_P^2 - 1)^2}{4x_P(x_P^2 + ax_P + 1)}, \tag{9}$$

which is well-defined and exception-free on odd-order subgroups. For example, when P is a point of order $\ell = 5$, we obtain the $d = 2$ relations

$$x_{[2]P} = [2]_x(x_P) \qquad \text{and} \qquad x_P = [2]_x(x_{[2]P}),$$

and when P is a point of order $\ell = 7$, we obtain the $d = 3$ relations

$$x_{[2]P} = [2]_x(x_P), \qquad x_{[3]P} = [2]_x(x_{[2]P}), \qquad \text{and} \qquad x_P = [2]_x(x_{[3]P}),$$

by exploiting that $x_{[3]P} = x_{-[3]P} = x_{[4]P}$ and $x_P = x_{-P} = x_{[6]P}$. In both cases, we rearranged (9) to make a the subject of each of the relations and took the average of the d equations to eliminate a in such a way that the substitution was symmetric in the x-coordinates of the d points of order ℓ. Furthermore, in both cases, the factorisation that paved the way to (6) was the result of trying several different lexicographical orderings of the variables and expressions to be eliminated under the above relations.

Remark 3. We point out that having one circular chain of relations like those exploited above does not hold in general. It always breaks down when ℓ is composite, and regularly breaks down when ℓ is prime. For example, when $\ell = 17$, we get two circular relations of length 4: one between the x-coordinates x_P, $x_{[2]P}$, $x_{[4]P}$, and $x_{[8]P}$, and another between $x_{[3]P}$, $x_{[5]P}$, $x_{[6]P}$, $x_{[7]P}$. To eliminate a in this case we could presumably write a as the 4-way average in both cases, then take the average of those two equations. Fortunately the pattern in (6) became visible at $\ell = 7$ before we needed to do this.

Remark 4. 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

[5] The nature of this observation is what we further exploit in the next section.

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, §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[6]. 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 Equation (6) into an analogue that, rather than writing the isogeny map in terms of the coordinates of the torsion points á la Vélu, instead writes it in terms of the coefficients of the polynomial defining the kernel subgroup á la Kohel [25, §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 [15,12,2,26,11], require separate functions for computing isogenous curves, i.e., $\texttt{iso_curve} : E \mapsto \phi(E)$, and for evaluating the isogeny at points, i.e., $\texttt{iso_point} : Q \mapsto \phi(Q)$. 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 `iso_curve` function with calls to `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.

Under the $\mathbf{x}$ map from Section 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

$$(a : 1) = (X_\alpha^2 + Z_\alpha^2 : \, - X_\alpha Z_\alpha). \tag{10}$$

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

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

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

so that the relation in (10) between 2-torsion coordinates 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 (10) at a cost[7] 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 (10) 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[8], 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[9]. In the next section we use the notation `a_from_alpha` to represent the function that performs this cheap parsing.

Remark 5. 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]

$$a = \frac{(1 - x_P x_Q - x_P x_{Q-P} - x_Q x_{Q-P})^2}{4 x_P x_Q x_{Q-P}} - x_P - x_Q - x_{Q-P}. \tag{11}$$

Thus, if we ever have 3 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 (11) 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 (11) 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 3 such x-coordinates that can be exploited during key generation, namely the 3 x-coordinates whose image under the isogeny forms (part of) the public key. During the shared secret computation, however, there will not always be 3 points available at each stage. Thus, we recommend that unless d is very large (so that the performance benefits of using (11) 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

[7] 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 $\mathbb{F}_{p^2}$.

[8] 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 $\alpha = i$.

[9] 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.

only equation we need to recall is (6), which we rewrite as

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

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

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

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

$$X' = X \cdot \Big(\prod_{i=1}^{d} \Big[(X - Z)(X_i + Z_i) + (X + Z)(X_i - Z_i) \Big] \Big)^2, \quad \text{and}$$
$$Z' = Z \cdot \Big(\prod_{i=1}^{d} \Big[(X - Z)(X_i + Z_i) - (X + Z)(X_i - Z_i) \Big] \Big)^2. \tag{12}$$

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 (12): 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`[10], we also need to input the modified curve constant $(\hat{A} : \hat{C}) = (a - 2 : 4)$.

[10] For many values of d (see Remark 3), *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`$((X_1 : Z_1), a)$ // 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** $((X_1 : Z_1), \ldots, (X_d : Z_d))$

Looking back at (12), 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 (12), we now present an algorithm for evaluating a single isogeny that takes as input the modified set of kernel point coordinates: `OddIsogeny` is given in Algorithm 3.

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** $(X' : Z')$

We are now in a position to present Algorithm 4, dubbed `SimultaneousOddIsogeny`, which is the main algorithm. 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 $\ker(\phi) = \langle P \rangle$.

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

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, Figure 2] or the *optimal* strategy [15, §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 Section 4 and the Montgomery ladder for computing the multiplication-by-ℓ map. We assume the use of the function `LADDER` as discussed in Section 4, where the Montgomery coefficient a is passed in projectively as $(\hat{A} : \hat{C}) = (a - 2 \colon 4)$; i.e.,

$$\texttt{LADDER} : \mathbb{P}^1 \times \mathbb{P}^1 \times \mathbb{Z} \to \mathbb{P}^1, \quad (\mathbf{x}(P), (\hat{A} : \hat{C}), \ell^z) \mapsto \mathbf{x}([\ell^z]P). \tag{13}$$

For ease of exposition, we adopt the multiplication-based approach [21, Figure 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.

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 $\alpha \neq 0$.
$(\mathbf{x}(Q_1), \ldots \mathbf{x}(Q_k)) = ((\mathbf{X}_1 : \mathbf{Z}_1), \ldots, (\mathbf{X}_k : \mathbf{Z}_k)) \in (\mathbb{P}^1)^k$, 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 $X_{\alpha'} \neq 0$.
$(\mathbf{x}(\phi(Q_1)), \ldots \mathbf{x}(\phi(Q_k))) = ((\mathbf{X}'_1 : \mathbf{Z}'_1), \ldots, (\mathbf{X}'_k : \mathbf{Z}'_k)) \in (\mathbb{P}^1)^k$

1 $((X_{\alpha'} : Z_{\alpha'}), (\mathbf{X}'_1 : \mathbf{Z}'_1), \ldots, (\mathbf{X}'_k : \mathbf{Z}'_k)) \leftarrow ((\alpha : 1), (\mathbf{X}_1 : \mathbf{Z}_1), \ldots, (\mathbf{X}_k : \mathbf{Z}_k))$ `// Initialise`
2 $(X_R : Z_R) \leftarrow (X_1 : Z_1)$ `// Initialise`
3 **for** $z = e - 1$ **downto** 0 **do**
4 $\quad (\hat{A} : \hat{C}) \leftarrow$ `a_from_alpha`$((X_{\alpha'} : Z_{\alpha'}))$ `// See Section 4`
5 $\quad (X_S \colon Z_S) \leftarrow$ `LADDER`$((X_R : Z_R), (\hat{A} : \hat{C}), \ell^z)$ `// See Equation (13)`
6 $\quad ((X_R : Z_R), (X_{\alpha'} : Z_{\alpha'}), (\mathbf{X}'_1 : \mathbf{Z}'_1), \ldots, (\mathbf{X}'_k : \mathbf{Z}'_k)) \leftarrow$ `// Algorithm 4`
$\quad$ `SimultaneousOddIsogeny`$((X_S : Z_S), (\hat{A} : \hat{C}), ((X_R : Z_R), (X_{\alpha'} : Z_{\alpha'}), (\mathbf{X}'_1 : \mathbf{Z}'_1), \ldots, (\mathbf{X}'_k : \mathbf{Z}'_k)))$
7 **end**
8 **return** $(X_{\alpha'} : Z_{\alpha'}), ((\mathbf{X}'_1 : \mathbf{Z}'_1), \ldots, (\mathbf{X}'_n : \mathbf{Z}'_n))$

In the notation of Section 2, let $E/\mathbb{F}_{p^2} \colon y^2 = x(x - \alpha)(x - 1/\alpha)$ 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

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

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

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

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

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 6 (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 (12) 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 $\mathbf{x}([i]P) = (X_i \colon Z_i)$ at a time and absorb its contribution to (12) immediately before replacing calling xADD to replace it by $\mathbf{x}([i+1]P) = (X_{i+1} \colon Z_{i+1})$, 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[11] accompanying [12,11]; this software uses the supersingular isogeny class containing the curve $E/\mathbb{F}_{p^2} \colon y^2 = x^3 + x$ 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 Section 2 or [15, §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 Section 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

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

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 Section 4; this is to view the relative performance of the simple SIDH loop in Algorithm 5 that computes evaluates each isogeny at one point during the main loop.

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

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.

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 Section 5), naturally these ratios become slightly more favourable for larger ℓ as n 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 Figure 1, and the approximate relative slowdown of using ℓ-isogenies within the SIDH framework is depicted on the right. An analogous version of Figure 1 for ℓ up to $\ell = 301$ is given in Figure 2. In both figures the cycle counts have been divided by n in order to give a cost per isogeny evaluation.

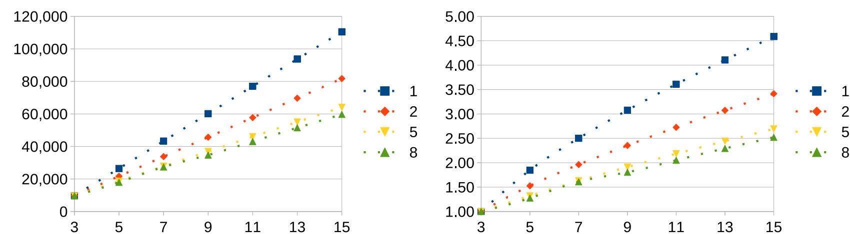

Fig. 1. Average cycle counts for `SimultaneousOddIsogeny` for different values of ℓ and n. Timing benchmarks were taken on an Intel Core i7-6500U Skylake processor running Ubuntu 14.04.5 LTS with TurboBoost disabled and all cores but one are switched-off. To obtain the executables, we used GNU-`gcc` 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 Figures 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 SIDH framework would incorporate the

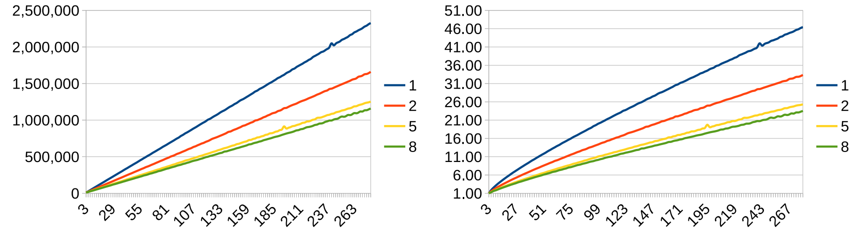

Fig. 2. Average cycle counts for `SimultaneousOddIsogeny` for different values of ℓ and n. Timing benchmarks were taken on an Intel Core i7-6500U Skylake processor running Ubuntu 14.04.5 LTS with TurboBoost disabled and all cores but one are switched-off. To obtain the executables, we used GNU-`gcc` 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.

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, §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 Figure 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.

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	-

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.

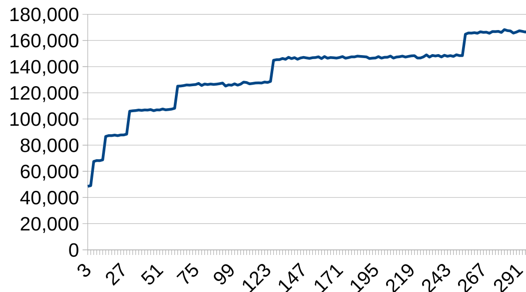

Fig. 3. Cycle counts chart for $[\ell](X:Z)$ using Montgomery ladder with projective inputs: $(X:Z)$ and $(A_{24}:C_{24})$. Timing benchmarks were taken on an Intel Core i7-6500U Skylake processor running Ubuntu 14.04.5 LTS with TurboBoost disabled and all cores but one are switched-off. To obtain the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

Implications. At a first glance, Table 1 and Figures 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, 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[12] $p = 4 \cdot 3^i 5^j - 1$ 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 option 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 be 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.

References

1. E. Alkim, L. Ducas, T. Pöppelmann, and P. Schwabe. Post-quantum key exchange - A new hope. In T. Holz and S. Savage, editors, *25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10-12, 2016.*, pages 327–343. USENIX Association, 2016. 2
2. R. Azarderakhsh, D. Jao, K. Kalach, B. Koziel, and C. Leonardi. Key compression for isogeny-based cryptosystems. In K. Emura, G. Hanaoka, and R. Zhang, editors, *Proceedings of the 3rd ACM*

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

International Workshop on ASIA Public-Key Cryptography, AsiaPKC@AsiaCCS, Xi'an, China, May 30 - June 03, 2016, pages 1–10. ACM, 2016. 2, 5, 8

3. D. J. Bernstein, P. Birkner, M. Joye, T. Lange, and C. Peters. Twisted Edwards curves. In S. Vaudenay, editor, *Progress in Cryptology - AFRICACRYPT 2008, First International Conference on Cryptology in Africa, Casablanca, Morocco, June 11-14, 2008. Proceedings*, volume 5023 of *Lecture Notes in Computer Science*, pages 389–405. Springer, 2008. 8
4. D. J. Bernstein, J. Buchmann, and E. Dahmen. *Post-quantum cryptography*. Springer Science & Business Media, 2009. 1
5. D. J. Bernstein, T. Chou, and P. Schwabe. McBits: Fast constant-time code-based cryptography. In G. Bertoni and J. Coron, editors, *Cryptographic Hardware and Embedded Systems - CHES 2013 - 15th International Workshop, Santa Barbara, CA, USA, August 20-23, 2013. Proceedings*, volume 8086 of *Lecture Notes in Computer Science*, pages 250–272. Springer, 2013. 1, 2
6. J. W. Bos, C. Costello, L. Ducas, I. Mironov, M. Naehrig, V. Nikolaenko, A. Raghunathan, and Douglas Stebila. Frodo: Take off the ring! Practical, quantum-secure key exchange from LWE. In E. R. Weippl, S. Katzenbeisser, C. Kruegel, A. C. Myers, and S. Halevi, editors, *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*, pages 1006–1018. ACM, 2016. 2
7. J. W. Bos, C. Costello, M. Naehrig, and D. Stebila. 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, May 17-21, 2015*, pages 553–570. IEEE Computer Society, 2015. 2
8. J. W. Bos and S. Friedberger. Fast arithmetic modulo $2^x p^y \pm 1$. Cryptology ePrint Archive, Report 2016/986, 2016. http://eprint.iacr.org/2016/986. 16
9. W. Castryck, S. Galbraith, and R. R. Farashahi. Efficient arithmetic on elliptic curves using a mixed Edwards-Montgomery representation. Cryptology ePrint Archive, Report 2008/218, 2008. http://eprint.iacr.org/2008/218. 8
10. T. Chou. Qcbits: Constant-time small-key code-based cryptography. In B. Gierlichs and A. Y. Poschmann, editors, *Cryptographic Hardware and Embedded Systems - CHES 2016 - 18th International Conference, Santa Barbara, CA, USA, August 17-19, 2016, Proceedings*, volume 9813 of *Lecture Notes in Computer Science*, pages 280–300. Springer, 2016. 1
11. C. Costello, D. Jao, P. Longa, M. Naehrig, J. Renes, and D. Urbanik. Efficient compression of SIDH public keys. In J. Coron and J. B. Nielsen, editors, *Advances in Cryptology - EUROCRYPT 2017 - 36th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Paris, France, April 30 - May 4, 2017, Proceedings, Part I*, volume 10210 of *Lecture Notes in Computer Science*, pages 679–706, 2017. 2, 5, 8, 9, 13
12. C. Costello, P. Longa, and M. Naehrig. Efficient algorithms for supersingular isogeny Diffie-Hellman. In M. Robshaw and J. Katz, editors, *Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part I*, volume 9814 of *Lecture Notes in Computer Science*, pages 572–601. Springer, 2016. 2, 5, 8, 9, 12, 13, 15, 19
13. J. Couveignes. Hard homogeneous spaces. Cryptology ePrint Archive, Report 2006/291, 2006. http://eprint.iacr.org/2006/291. 2
14. W. Diffie and M. E. Hellman. New directions in cryptography. *IEEE Trans. Information Theory*, 22(6):644–654, 1976. 4
15. L. De Feo, D. Jao, and J. Plût. Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies. *J. Mathematical Cryptology*, 8(3):209–247, 2014. 2, 3, 5, 7, 8, 12, 13
16. S. Fluhrer. Cryptanalysis of ring-lwe based key exchange with key share reuse. Cryptology ePrint Archive, Report 2016/085, 2016. http://eprint.iacr.org/2016/085. 2
17. E. Fujisaki and T. Okamoto. How to enhance the security of public-key encryption at minimum cost. In H. Imai and Y. Zheng, editors, *Public Key Cryptography, Second International Workshop on Practice and Theory in Public Key Cryptography, PKC '99, Kamakura, Japan, March 1-3, 1999, Proceedings*, volume 1560 of *Lecture Notes in Computer Science*, pages 53–68. Springer, 1999. 2
18. S. D. Galbraith. *Mathematics of public key cryptography*. Cambridge University Press, 2012. 3
19. S. D. Galbraith, C. Petit, B. Shani, and Y. B. Ti. On the security of supersingular isogeny cryptosystems. In J. H. Cheon and T. Takagi, editors, *Advances in Cryptology - ASIACRYPT 2016 - 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part I*, volume 10031 of *Lecture Notes in Computer Science*, pages 63–91, 2016. 2
20. J. Hoffstein, J. Pipher, and J. H. Silverman. NTRU: A ring-based public key cryptosystem. In J. Buhler, editor, *ANTS-III*, volume 1423 of *LNCS*, pages 267–288. Springer, 1998. 1

21. D. Jao and L. De Feo. Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies. In B. Yang, editor, *Post-Quantum Cryptography - 4th International Workshop, PQCrypto 2011, Taipei, Taiwan, November 29 - December 2, 2011. Proceedings*, volume 7071 of *Lecture Notes in Computer Science*, pages 19–34. Springer, 2011. 2, 4, 5, 12
22. D. Kirkwood, B. C. Lackey, J. McVey, M. Motley, J. A. Solinas, and D. Tuller. Failure is not an option: Standardization issues for post-quantum key agreement. Talk at NIST workshop on Cybersecurity in a Post-Quantum World: `http://www.nist.gov/itl/csd/ct/post-quantum-crypto-workshop-2015.cfm`, April, 2015. 2
23. K. Kobara and H. Imai. Semantically secure McEliece public-key cryptosystems-conversions for McEliece PKC. In K. Kim, editor, *Public Key Cryptography, 4th International Workshop on Practice and Theory in Public Key Cryptography, PKC 2001, Cheju Island, Korea, February 13-15, 2001, Proceedings*, volume 1992 of *Lecture Notes in Computer Science*, pages 19–35. Springer, 2001. 2
24. P. C. Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In N. Koblitz, editor, *Advances in Cryptology - CRYPTO '96, 16th Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 1996, Proceedings*, volume 1109 of *Lecture Notes in Computer Science*, pages 104–113. Springer, 1996. 5
25. D. R. Kohel. *Endomorphism rings of elliptic curves over finite fields.* PhD thesis, University of California, Berkeley, 1996. 8
26. B. Koziel, R. Azarderakhsh, M. M. Kermani, and D. Jao. Post-quantum cryptography on FPGA based on isogenies on elliptic curves. *IEEE Trans. on Circuits and Systems*, 64-I(1):86–99, 2017. 2, 5, 8
27. V. Lyubashevsky, C. Peikert, and O. Regev. On ideal lattices and learning with errors over rings. *J. ACM*, 60(6):43:1–43:35, 2013. 2
28. R. J. McEliece. A public-key cryptosystem based on algebraic coding theory. *Coding Thv*, 4244:114–116, 1978. 1
29. R. Misoczki, J. Tillich, N. Sendrier, and P. S. L. M. Barreto. MDPC-McEliece: New McEliece variants from moderate density parity-check codes. In *Proceedings of the 2013 IEEE International Symposium on Information Theory, Istanbul, Turkey, July 7-12, 2013*, pages 2069–2073. IEEE, 2013. 1
30. P. L. Montgomery. Speeding the Pollard and elliptic curve methods of factorization. *Mathematics of computation*, 48(177):243–264, 1987. 3, 4, 9, 10, 12
31. D. Moody and D. Shumow. Analogues of Vélu's formulas for isogenies on alternate models of elliptic curves. *Math. Comput.*, 85(300):1929–1951, 2016. 3, 5, 6, 7, 8
32. M. Mosca. Cybersecurity in an era with quantum computers: will we be ready? Cryptology ePrint Archive, Report 2015/1075, 2015. `http://eprint.iacr.org/2015/1075`. 1
33. C. Peikert. Lattice cryptography for the Internet. In *Post-Quantum Cryptography - 6th International Workshop, PQCrypto 2014, Waterloo, ON, Canada, October 1-3, 2014. Proceedings*, pages 197–219, 2014. 2
34. O. Regev. On lattices, learning with errors, random linear codes, and cryptography. In H. N. Gabow and R. Fagin, editors, *Proceedings of the 37th Annual ACM Symposium on Theory of Computing, Baltimore, MD, USA, May 22-24, 2005*, pages 84–93. ACM, 2005. 2
35. A. Rostovtsev and A. Stolbunov. Public-key cryptosystem based on isogenies. Cryptology ePrint Archive, Report 2006/145, 2006. `http://eprint.iacr.org/`. 2
36. P. W. Shor. Algorithms for quantum computation: Discrete logarithms and factoring. In *Foundations of Computer Science, 1994 Proceedings., 35th Annual Symposium on*, pages 124–134. IEEE, 1994. 1
37. D. Stebila and M. Mosca. 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`. 1
38. A. Stolbunov. *Cryptographic Schemes Based on Isogenies.* PhD thesis, Norwegian University of Science and Technology, 2012. 2
39. The National Institute of Standards and Technology (NIST). Submission requirements and evaluation criteria for the post-quantum cryptography standardization process, December, 2016. 1
40. J. Vélu. Isogénies entre courbes elliptiques. *CR Acad. Sci. Paris Sér. AB*, 273:A238–A241, 1971. 5, 6
41. Y. Yoo, R. Azarderakhsh, A. Jalali, D. Jao, and V. Soukharev. A post-quantum digital signature scheme based on supersingular isogenies. *To appear in Financial Cryptography and Data Security*, 2017. Preprint at `http://eprint.iacr.org/2017/186`. 3

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.

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	8**M**+4**S**+8**a**	6**M**+2**S**+2**a**	3**M**+3**S**+8**a**	9**M**+1**S**+6**a**	5**S**+7**a**
This work	9**M**+2**S**+17**a** 7**M**+5**S**+11**a**	4**M**+2**S**+4**a**	2**M**+3**S**+14**a**	6**M**+2**S**+6**a**	4**S**+4**a**

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_curve operation

$$(A'_{24} : C'_{24}) = \big((X_3 + Z_3)(Z_3 - 3X_3)^3 : 16X_3Z_3^3\big)$$

takes 2**M**+3**S**+14**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:

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

The `4_iso_curve` operation

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

takes 4**S**+4**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

$$\begin{aligned}(X' : Z') = \big(&X\left(2X_4Z_4Z - (X_4^2 + Z_4^2)X\right)(X_4X - Z_4Z)^2 : \\ &Z\left(2X_4Z_4X - (X_4^2 + Z_4^2)Z\right)(Z_4X - X_4Z)^2\big)\end{aligned}$$

takes 6**M**+2**S**+6**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

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

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:

$R_1 = X - Z$, $R_2 = R_1^2$, $R_3 = X + Z$, $R_4 = R_3^2$, $R_5 = R_4 + R_2$, $R_6 = R_2 - R_4$, $R_7 = R_4 \cdot C_1$, $R_8 = R_2 \cdot A_{24}$, $R_4 = R_8 + R_7$, $R_2 = R_7 - R_8$, $R_4 = R_4 \cdot R_6$, $R_5 = R_2 \cdot R_5$, $R_2 = R_2 \cdot R_6$, $R_2 = 2R_2$, $R_6 = R_4 + R_5$, $R_5 = R_4 - R_5$, $R_4 = R_6 + R_2$, $R_6 = R_6 - R_2$, $R_4 = R_4 \cdot R_6$, $R_6 = R_2 \cdot R_5$, $R_6 = 2R_6$, $R_5 = R_4 - R_6$, $R_4 = R_4 + R_6$, $R_2 = R_4 \cdot R_3$, $R_1 = R_1 \cdot R_5$, $X_{out} = R_2 + R_1$, $Z_{out} = R_2 - R_1$.

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**M**+6**S**+14**a**, assuming that $K_1 = A_{24} + C_{24}$ is cached.

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

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.

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