Efficient algorithms for supersingular isogeny Diffie-Hellman

Craig Costello, Patrick Longa, and Michael Naehrig

Microsoft Research, USA

Abstract. We propose a new suite of algorithms that significantly improve the performance of supersingular isogeny Diffie-Hellman (SIDH) key exchange. Subsequently, we present a full-fledged implementation of SIDH that is geared towards the 128-bit quantum and 192-bit classical security levels. Our library is the first constant-time SIDH implementation and is more than 2.5 times faster than the previous best (non-constant-time) SIDH software. The high speeds in this paper are driven by compact, inversion-free point and isogeny arithmetic and fast SIDH-tailored field arithmetic: on an Intel Haswell processor, generating ephemeral public keys takes 51 million cycles for Alice and 59 million cycles for Bob while computing the shared secret takes 47 million and 57 million cycles, respectively. The size of public keys is only 751 bytes, which is significantly smaller than most of the popular post-quantum key exchange alternatives. Ultimately, the size and speed of our software illustrates the strong potential of SIDH as a post-quantum key exchange candidate and we hope that these results encourage a wider cryptanalytic effort.

Keywords: Post-quantum cryptography, Diffie-Hellman key exchange, supersingular elliptic curves, isogenies, SIDH.

1 Introduction

Post-quantum cryptography. The prospect of a large scale quantum computer that is capable of implementing Shor's algorithm [50] has given rise to the field of post-quantum cryptography (PQC). Its goal is to develop and ultimately deploy cryptographic primitives that resist cryptanalysis by both classical *and* quantum computers. Recent developments in quantum computing (see, e.g., [18, 40, 29]) have helped catalyze government and corporate action in this arena. For example, in April 2015, the National Institute of Standards and Technology (NIST) held a "Workshop on Cybersecurity in a Post-Quantum World", reaching out to academia and industry to discuss potential future standardization of PQC. Later, in August 2015, the National Security Agency (NSA) released a major policy statement that announced plans to "transition to quantum resistant algorithms in the not too distant future"[41]. In February 2016, NIST published a draft "Report on Post-Quantum Cryptography" [13], which emphasizes the need to start working towards the deployment of post-quantum cryptography in our information security systems, and outlines NIST's plans to "initiate a standardization effort in post-quantum cryptography".

In terms of public-key PQC, there are four well-known and commonly cited classes of cryptographic primitives that are believed to remain secure in the presence of a quantum computer: code-based cryptography, lattice-based cryptography, hash-based cryptography, and multivariate cryptography. Specific examples for each of these are McEliece's code-based encryption scheme [35]; Hoffstein, Pipher and Silverman's lattice-based encryption scheme "NTRU" [27]; Merkle's hash-tree signatures [36]; and Patarin's "HFE^{v-}" signature scheme [44]. A positive trait shared by all of these examples is a resistance to decades of attempted classical and quantum cryptanalysis which has inspired widespread confidence in their suitability as a post-quantum primitive. However, most of these examples also share the trait of having enormous public key and/or signature sizes, particularly when compared to traditional primitives based on the hardness of integer factorization or (elliptic curve) discrete logarithm computation.

Supersingular isogeny Diffie-Hellman. In this paper, we study a different primitive that does not fall into any of the above classes, but is currently believed to offer post-quantum resistance: the supersingular isogeny Diffie-Hellman (SIDH) key exchange protocol proposed by Jao and De Feo in 2011 [28]. The SIDH key exchange protocol is more than a decade younger than all of the above schemes, so its security is yet to withstand the tests of time and of a wide cryptanalytic effort. Nevertheless, the current picture of its security properties looks promising. The best known classical and quantum attacks against the underlying problem are both exponential in the size of the underlying finite field, and their complexities make current SIDH key sizes significantly smaller than their post-quantum key exchange and/or encryption counterparts[1].

Our contributions. We present a full-fledged, high-speed implementation of (unauthenticated) ephemeral SIDH that currently provides 128 bits of quantum security and 192 bits of classical security. This implementation uses 48-byte private keys to produce 751-byte ephemeral Diffie-Hellman public keys, and is currently written almost entirely in C with only a limited set of functions written in assembly. To our knowledge, our library (see [16]) presents the first SIDH software that runs in *constant-time*, i.e., that is designed to resist timing [32] and cache timing [43] attacks. On x64 platforms, our implementation runs more than 2.5 times faster than the (previously fastest) implementation of SIDH by Azarderakhsh, Fishbein and Jao [2]. Note that this performance comparison does not take into account the fact that the implementation from [2] is not protected against timing attacks.

The main technical contributions that lead to these improvements are:

- *Projective curve coefficients:* a widely-deployed technique in traditional ECC involves avoiding inversions by working with elliptic curve points in projective space. Following Jao and De Feo [28], we also employ this technique to work efficiently with points in $\mathbb{P}^1$ by making use of the fast arithmetic associated with the Kummer varieties of Montgomery curves. A crucial difference in this work, however, is that we also work projectively with the curve coefficients; unlike traditional ECC where the curve is fixed, every SIDH key exchange requires computations on many different isogenous curves. In Section 3 we show that the Montgomery model also allows all of the necessary isogeny arithmetic to be performed efficiently in $\mathbb{P}^1$. This gives rise to more compact algorithms, significantly simplifies the overall computation, and in particular means that only one inversion is required during each round of the SIDH protocol. The observations in Section 3 ultimately show that using the Kummer varieties of Montgomery curves gives a twofold advantage in SIDH: both point arithmetic *and* isogeny arithmetic amount to simple computations in $\mathbb{P}^1$.
- *Prime selection and tailor-made Montgomery multiplication:* we select a prime with form $p = \ell_A^{e_A} \ell_B^{e_B} f - 1$, where $\ell_A = 2$, $\ell_B = 3$, and the bit lengths of 2^{e_A} and 3^{e_B} are slightly smaller than a multiple of 64. This supports efficient arithmetic on a wide range of platforms and allows access to a large variety of optimizations such as the efficient use of vector instructions, Karatsuba multiplication, and lazy reduction. Moreover, it is well-known that primes of a special form can lead to faster algorithms for computing modular arithmetic in comparison with general-purpose algorithms. In this work, we note the special shape of these *SIDH-friendly* primes and modify the popular Montgomery multiplication algorithm to speed up modular arithmetic.
- *Ground field scalar multiplications for key generation:* secure key generation in the SIDH protocol requires the definition of two independent cyclic subgroups of a fixed order (see Section 2). Jao and De Feo [28, §4.1] propose that generators of these two groups can be computed by multiplying random curve points by an appropriate cofactor, and that their linear independence can be checked via the Weil pairing. In Section 4 we employ a well-known technique from the pairing literature [48, §5] to work with two advantageous choices of torsion subgroups: the *base-field* and *trace-zero* subgroups. These choices allow the initial

[1] An exception here is NTRUEncrypt [27], which has comparable public key sizes – see https://github.com/NTRUOpenSourceProject/ntru-crypto.

scalar multiplications that are required during key generation to be performed entirely over the base field. While these scalar multiplications only constitute a small fraction of the overall key generation time, and therefore the overall speedup from this technique is only moderate, a more visible benefit is the significant decrease in the size of the public parameters – see Section 6. We discuss possible security implications of this choice in Section 4.

Several of the above choices not only aid efficiency, but also the overall simplicity and compactness of the SIDH scheme. Choosing to unify points with their inverses and to unify Montgomery curves with their quadratic twists (see Section 3) effectively compresses the elements that are sent over the wire, i.e., the public keys, by a factor of two. Moreover, there is no decompression required and, in particular, our software never requires the computation of square roots[2].

The timings we present in Section 7 reveal that high-security SIDH key exchange is more efficient than it was previously known to be. Moreover, our constant-time software shows that, if confidence in the security of SIDH warrants real-world deployment in the future, the same level of side-channel protection can be achieved in the SIDH setting as in traditional number-theoretic schemes. We therefore hope that this paper encourages a wider cryptanalytic effort on the problems underlying the security of SIDH (see Section 2). Moreover, even if cryptanalytic improvements are made in the future, the huge difference between current SIDH key sizes and those of other PQC primitives suggest that the problem could remain of interest to practitioners. So long as the best known attacks remain exponential with a reasonable exponent (see the discussion below), it is reasonable to suggest that elliptic curves could offer the same benefit in post-quantum cryptography that they did in classical cryptography.

Beyond the efficiency improvements above, we present several techniques that help to bridge the gap between the theoretical SIDH scheme in [28] and its real-world deployment. Of particular importance are the contributions discussed in the following two paragraphs.

A strong ECDH+SIDH hybrid. Given the uncertainty surrounding the arrival date of large-scale quantum computers (as well as the time it takes for new primitives to be thoroughly cryptanalyzed, standardized and deployed), many real-world cryptographers are hastily pushing for deployment of post-quantum primitives sooner rather than later. Subsequently, a proposal that is gaining popularity in the PQC community is the deployment of *hybrid* schemes, i.e., schemes where a long-standing classically-secure primitive $\mathcal{P}$ is partnered alongside a newer post-quantum candidate $\mathcal{Q}$ (cf. [7]). The simple reasoning here is that, even if further cryptanalysis weakens $\mathcal{Q}$'s resistance to classical computers, the hybrid scheme $\mathcal{P} + \mathcal{Q}$ is likely to remain classically secure; conversely, $\mathcal{P}$'s presumed weakness against a quantum computer does not affect the post-quantum security of $\mathcal{P} + \mathcal{Q}$. Taking such a prudent measure in the case of SIDH, which is much newer than other post-quantum primitives, seems especially wise. In Section 8 we present a possibility to partner SIDH public keys alongside traditional elliptic curve Diffie-Hellman (ECDH) public keys that are extremely strong. In particular, while our proposed SIDH parameters respectively offer around 128 and 192 bits of security against the best known quantum and classical attacks, the proposed hybrid offers around 384 bits of classical security based on the elliptic curve discrete logarithm problem (ECDLP). While this might seem like overkill[3], we show that this partnering is a very natural choice and comes at a relatively small cost: compared to a standalone SIDH, the size of the public keys and the overall runtime in our SIDH+ECDH hybrid increase by no more than 17% and 13%, respectively, and there is almost no additional code required to include ECDH in the scheme.

Full public key validation. The security of unauthenticated ephemeral key exchange is modeled using passive adversaries, in which case we can assume that both parties' public keys are honestly generated. As was pointed out in April 2015 by a group at the NSA [30], in static key exchange

[2] Square roots in the types of fields used in SIDH are particularly annoying and require multiple field exponentiations [49, §3.3].

[3] For example, this is double the bit-size of the 192-bit ECDH currently recommended by the NSA [41] for "up to TOP SECRET" protection.

when private keys are reused, validating public keys in the case of isogeny-based cryptography becomes both necessary and non-trivial. In Section 9 we detail the validation of the public keys used in our scheme, and in particular show how to achieve full validation efficiently in our compact framework. To our knowledge this is the first time public key validation has been described in the case of SIDH.

SIDH history and security. Beginning with an unpublished preprint with Rostovtsev in early 2006 [46], and then in a series of Russian papers that culminated in his thesis [52][4], Stolbunov proposed a Diffie-Hellman-like cryptosystem based on the difficulty of computing isogenies between ordinary (i.e., non-supersingular) elliptic curves. The best algorithm to solve this problem on a classical computer runs in exponential time and is due to Galbraith and Stolbunov [24][5]. In late 2010, however, Childs, Jao and Soukharev [14] gave a quantum algorithm that computes isogenies between ordinary curves in subexponential time, assuming the Generalized Riemann Hypothesis (GRH). Subsequently, in late 2011, Jao and De Feo [28] put forward SIDH, which is instead based on the difficulty of computing isogenies between supersingular elliptic curves. This problem is immune to the quantum attack in [14], since this attack crucially relies on the endomorphism ring being commutative, which is not the case for a supersingular curve whose endomorphism ring is isomorphic to an order in a quaternion algebra [51, §V.3.1].

Given two isogenous supersingular elliptic curves defined over a field of characteristic p, the *general* supersingular isogeny problem is to construct an isogeny between them. The best known classical attack on this problem is due to Delfs and Galbraith [17] and requires $\tilde{O}(p^{1/2})$ bit operations, while the best known quantum attack is due to Biasse, Jao and Sankar [8] and requires $\tilde{O}(p^{1/4})$ bit operations. The problems underlying SIDH (see Section 2) are not general in that the degree of the isogeny, which is smooth and in $O(\sqrt{p})$, is known and public. As is discussed by De Feo, Jao and Plût [19, §5.1][6], this specialized problem can be viewed as an instance of the *claw problem*, and the optimal asymptotic classical and quantum complexities for the claw problem are known to be $O(p^{1/4})$ and $O(p^{1/6})$, respectively [59, 54]. Currently, this approach yields the best known classical and quantum attacks against SIDH.

Organization. In Section 2 we recall the key concepts from [19] that are needed in SIDH. In Section 3 we show that all isogeny and point computations can be performed in $\mathbb{P}^1$; here we derive all of the lower-level functions that are called during the key generation and shared secret operations. In Section 4 we fix the underlying isogeny class used in our software, describe the high-level key exchange operations, and discuss other implementation choices. In Section 5 we detail the special field arithmetic that is tailored towards our chosen prime (as well as many other well-chosen SIDH-friendly primes).

We give a summary of the scheme in Section 6 and present performance results of our implementation in Section 7. In Section 8 we describe our proposal for a strong hybrid key exchange scheme that combines classical ECDH with post-quantum SIDH. Section 9 shows how to efficiently validate SIDH public keys in static key exchange settings, and Section 10 discusses (further) public key compression. We conclude the paper in Section 11.

To promote future implementations of SIDH, we have endeavored to make this paper as self-contained as possible. Essentially, all functions that are needed to implement SIDH are fully described in Section 3. High level functions are given in explicit form in the appendix. All other details can be found in the released code [16].

[4] See [52, §1.3] for a history of isogeny-based cryptographic proposals prior to Stolbunov's work.

[5] This is based on the algorithm for constructing isogenies by Galbraith, Hess and Smart [21, §3], which itself is an extension of the original algorithm by Galbraith [20].

[6] This is an extended version of the original SIDH paper by Jao and De Feo [28].

2 Diffie-Hellman key exchange from supersingular elliptic curve isogenies

This section sets the stage by introducing notation, giving some basic properties of torsion subgroups and isogenies, and recalling the supersingular isogeny Diffie-Hellman key exchange protocol. This is all described in a similar fashion by De Feo, Jao and Plût in [19, §2].

Smooth order supersingular elliptic curves. SIDH uses isogeny classes of supersingular elliptic curves with smooth orders so that rational isogenies of exponentially large (but smooth) degree can be computed efficiently as a composition of low degree isogenies. Fix two small prime numbers ℓ_A and ℓ_B, an integer cofactor f, and let p be a prime of the form $p = \ell_A^{e_A}\ell_B^{e_B}f \pm 1$. It is then easy to construct a supersingular elliptic curve E defined over $\mathbb{F}_{p^2}$ of order $(\ell_A^{e_A}\ell_B^{e_B}f)^2$ [11].

For $\ell \in \{\ell_A, \ell_B\}$ and $e \in \{e_A, e_B\}$ the corresponding exponent, we have that the full ℓ^e-torsion group on E is defined over $\mathbb{F}_{p^2}$, i.e. $E[\ell^e] \subseteq E(\mathbb{F}_{p^2})$. Since ℓ is coprime to p, $E[\ell^e] \cong (\mathbb{Z}/\ell^e\mathbb{Z}) \times (\mathbb{Z}/\ell^e\mathbb{Z})$ [51, III.6.4]. Let $P, Q \in E[\ell^e]$ be two points that generate $E[\ell^e]$ such that the above isomorphism is given by $(\mathbb{Z}/\ell^e\mathbb{Z}) \times (\mathbb{Z}/\ell^e\mathbb{Z}) \to E[\ell^e],\ (m,n) \mapsto [m]P + [n]Q$. Roughly speaking, the SIDH secret keys are degree ℓ^e isogenies of the base curve E, which are in one-to-one correspondence with the cyclic subgroups of order ℓ^e that form their kernels. A point $[m]P + [n]Q$ has full order ℓ^e if and only if at least either m or n are not divisible by ℓ. There are $\ell^{2e-2}(\ell^2 - 1)$ such points. Since distinct cyclic subgroups only intersect in points of order less than ℓ^e and all full-order points in a single subgroup are coprime multiples of one such point, it follows that there are $\ell^{e-1}(\ell + 1)$ distinct cyclic subgroups of order ℓ^e.

Computing large degree isogenies. Given a cyclic subgroup $\langle R \rangle \subseteq E[\ell^e]$ of order ℓ^e, there is a unique isogeny ϕ_R of degree ℓ^e, defined over $\mathbb{F}_{p^2}$ with kernel $\langle R \rangle$ [51, III.4.12], mapping E to an isogenous elliptic curve $E/\langle R \rangle$. The isogeny ϕ_R can be computed as the composition of e isogenies of degree ℓ which in turn can be computed by using Vélu's formulas [56]. As described in [19, §4.2.2], we can start with $E_0 := E$ and $R_0 := R$ and then iteratively compute $E_{i+1} = E_i/\langle [\ell^{e-i-1}]R_i \rangle$ for $0 \le i < e$ as follows. Each iteration computes the degree-ℓ isogeny $\phi_i : E_i \to E_{i+1}$ whose kernel is the cyclic group $\langle [\ell^{e-i-1}]R_i \rangle$ of order ℓ, before applying the isogeny to compute $R_{i+1} = \phi_i(R_i)$. The point R_i is an (ℓ^{e-i})-torsion point and so $[\ell^{e-i-1}]R_i$ has order ℓ. Thus, the composition $\phi_R = \phi_{e-1} \circ \cdots \circ \phi_0$ has degree ℓ^e, which together with $(\phi_{e-1} \circ \cdots \circ \phi_0)(R) = R_e = \mathcal{O}$ shows that $\ker(\phi_R) = \langle R \rangle$, and therefore that $\phi = \phi_{e-1} \circ \cdots \circ \phi_0$.

There are two obvious ways of computing ϕ using the above decomposition. One of them follows directly from the description above: in each iteration, one first computes the scalar multiplication $[\ell^{e-i-1}]R_i$ to obtain a point of order ℓ, then uses Vélu's formulas to compute ϕ_i, and evaluates it at R_i to obtain the next point R_{i+1}. Jao and De Feo [28, Figure 2] call this the multiplication-based strategy because it is dominated by the number of scalar multiplications by ℓ that are needed to obtain the ℓ-torsion points. The second obvious approach is called the isogeny-based method [28, Figure 2] because it is dominated by the number of isogeny evaluations. It requires only one loop of scalar-multiplications that stores all ℓ-multiples of R, i.e., all intermediate results $Q_i = [\ell^i]R$ for $0 \le i < e$. The point Q_{e-1} has order ℓ and can be used to obtain the isogeny ϕ_0 as above. One then replaces all Q_i for $0 \le i \le (e-2)$ by $\phi_0(Q_i)$. At this point Q_{e-2} has order ℓ and is used to obtain ϕ_1. This is repeated until one obtains ϕ_{e-1} and hence the composition ϕ.

De Feo, Jao and Plût [19, §4.2.2] demonstrate that both of these methods are rather wasteful and that there is a much more efficient way to schedule the multiplications-by-ℓ and ℓ-isogeny evaluations. Section 4 presents more details about this for our specific parameter choice.

SIDH key exchange. This paragraph recalls the SIDH key exchange protocol from [19, §3.2]. The public parameters are the supersingular curve $E_0/\mathbb{F}_{p^2}$ whose group order is $(\ell_A^{e_A}\ell_B^{e_B}f)^2$, two independent points P_A and Q_A that generate $E_0[\ell_A^{e_A}]$, and two independent points P_B and Q_B that generate $E_0[\ell_B^{e_B}]$. To compute her public key, Alice chooses two secret integers $m_A, n_A \in \mathbb{Z}/\ell_A^{e_A}\mathbb{Z}$, not both divisible by ℓ_A, such that $R_A = [m_A]P_A + [n_A]Q_A$ has order $\ell_A^{e_A}$. Her secret key is computed as the degree $\ell_A^{e_A}$ isogeny $\phi_A\colon E_0 \to E_A$ whose kernel is R_A, and her public key is the isogenous curve E_A together with the image points $\phi_A(P_B)$ and $\phi_A(Q_B)$. Similarly, Bob chooses two secret integers $m_B, n_B \in \mathbb{Z}/\ell_B^{e_B}\mathbb{Z}$, not both divisible by ℓ_B, such that $R_B = [m_B]P_B + [n_B]Q_B$ has order $\ell_B^{e_B}$. He then computes his secret key as the degree $\ell_B^{e_B}$ isogeny $\phi_B\colon E_0 \to E_B$ whose kernel is R_B, and his public key is E_B together with $\phi_B(P_A)$ and $\phi_B(Q_A)$. To compute the shared secret, Alice uses her secret integers and Bob's public key to compute the degree $\ell_A^{e_A}$ isogeny $\phi'_A\colon E_B \to E_{BA}$ whose kernel is the point $[m_A]\phi_B(P_A) + [n_A]\phi_B(Q_A) = \phi_B([m_A]P_A + [n_A]Q_A) = \phi_B(R_A)$. Similarly, Bob uses his secret integers and Alice's public key to compute the degree $\ell_B^{e_B}$ isogeny $\phi'_B\colon E_B \to E_{AB}$ whose kernel is the point $[m_B]\phi_A(P_B) + [n_B]\phi_A(Q_B) = \phi_A(R_B)$. It follows that E_{BA} and E_{AB} are isomorphic, so Alice and Bob can compute a shared secret as the common j-invariant $j(E_{BA}) = j(E_{AB})$.

We note that the number of possibilities for Alice to choose her secret isogeny is equal to the number of possible distinct kernels, which is $\ell_A^{e_A-1}(\ell_A + 1)$, and analogously the number of possible choices for Bob is $\ell_B^{e_B-1}(\ell_B + 1)$.

Security under SSDDH. In [19, §5], De Feo, Jao and Plût give a number of computational problems related to SIDH and discuss their complexity. In [19, §6], they prove that SIDH is *session-key secure* in the authenticated-links adversarial model of Canneti and Krawczyk [12] under the Supersingular Decision Diffie-Hellman (SSDDH) problem, which we recall as follows. With the public parameters as above, one is given a tuple sampled with probability 1/2 from either one of the following two distributions:

– $(E_A, E_B, \phi_A(P_B), \phi_A(Q_B), \phi_B(P_A), \phi_B(Q_A), E_{AB})$, where
$$E_{AB} \cong E_0/\big\langle [m_A]P_A + [n_A]Q_A, [m_B]P_B + [n_B]Q_B \big\rangle.$$

– $(E_A, E_B, \phi_A(P_B), \phi_A(Q_B), \phi_B(P_A), \phi_B(Q_A), E_C)$, where
$$E_C \cong E_0/\big\langle [m'_A]P_A + [n'_A]Q_A, [m'_B]P_B + [n'_B]Q_B \big\rangle,$$

and the values m'_A, n'_A, m'_B and n'_B are chosen randomly from the same respective distributions as m_A, n_A, m_B and n_B. The SSDDH problem is to determine from which distribution the tuple is sampled.

We note that the Canneti-Krawczyk model assumes perfectly authenticated links which effectively forces adversaries to be passive eavesdroppers. We return to this discussion in Section 9.

3 Projective points and projective curve coefficients

In this section we present one of our main technical contributions by showing that, just as the Montgomery form allows point arithmetic to be carried out efficiently in $\mathbb{P}^1$, in the context of SIDH it also allows isogeny arithmetic to be carried out in $\mathbb{P}^1$. This gives rise to fast, inversion-free point-and-isogeny operations that significantly boost the performance of SIDH. In comparison to the software[7] accompanying [19] that computes at least one inversion per isogeny computation, and therefore $O(\ell)$ inversions per round of the protocol, our software only requires one inversion per round. We describe all of the point and isogeny operations that are needed and conclude the section with a table summarizing their operation counts. For all of the explicit formulas in optimized two-operand form, see `SIDH.mag` in [16].

[7] See https://github.com/defeo/ss-isogeny-software/.

Montgomery curves. Over a field K, a Montgomery curve [39] is defined by the two constants $(a, b) \in \mathbb{A}^2(K)$ as $E_{(a,b)} \colon by^2 = x^3 + ax^2 + x$. Unlike traditional ECC, in this work the defining curve does not stay fixed, but changes as we *move around* an isogeny class. As we discuss further below, it is therefore convenient to work projectively both with points on curves *and* with the curve coefficients themselves. Let $(A \colon B \colon C) \in \mathbb{P}^2(K)$ with $C \in \bar{K}^{\times}$ be such that $a = A/C$ and $b = B/C$. Then $E_{(a,b)}$ can alternatively be written as

$$E_{(A\colon B\colon C)} \colon By^2 = Cx^3 + Ax^2 + Cx.$$

The K-rational points on $E_{(a,b)}$ or $E_{(A\colon B\colon C)}$ are contained in $\mathbb{P}^2(K)$, so as usual we use the notation $(X \colon Y \colon Z) \in \mathbb{P}^2(K)$ with $Z \neq 0$ to represent all points $(x, y) = (X/Z, Y/Z)$ in $\mathbb{A}^2(K)$, and the point at infinity is $\mathcal{O} = (0 \colon 1 \colon 0)$.

The j-invariants of the curves given by these models are

$$j(E_{a,b}) = \frac{256(a^2-3)^3}{a^2-4} \qquad \text{and} \qquad j(E_{(A\colon B\colon C)}) = \frac{256(A^2-3C^2)^3}{C^4(A^2-4C^2)}. \tag{1}$$

Kummer varieties and points in $\mathbb{P}^1$. Following [39], viewing the x-line $\mathbb{P}^1$ as the Kummer variety of $E_{(a,b)}$ allows for particularly efficient arithmetic in $E_{(a,b)}/\langle \pm 1 \rangle \cong \mathbb{P}^1$. Let

$$x \colon \; E_{(a,b)} \setminus \{\mathcal{O}\} \to \mathbb{P}^1, \; (X \colon Y \colon Z) \mapsto (X \colon Z).$$

For the points $P, Q \in E_{(a,b)} \setminus \{\mathcal{O}\}$ and $m \in \mathbb{Z}$, Montgomery [39] gave efficient formulas for computing the doubling function $\texttt{xDBL} \colon (x(P), a) \mapsto x([2]P)$, the function $\texttt{xADD} \colon (x(P), x(Q), x(Q-P)) \mapsto x(Q+P)$ for *differential additions*, and the function $\texttt{xDBLADD} \colon (x(P), x(Q), x(Q-P), a) \mapsto (x([2]P), x(Q-P))$ for the merging of the two. These are all ingredients in the Montgomery ladder function to compute the $\mathbb{Z}$-action on $E_{(a,b)}/\langle \pm 1 \rangle \cong \mathbb{P}^1$, i.e., $\texttt{LADDER} \colon (x(P), a, m) \mapsto x([m]P)$.

We note that the xADD function works identically for $E_{(a,b)}$ and $E_{(A\colon B\colon C)}$, while the other functions on $E_{(a,b)}$ that involve a can be trivially modified to work on $E_{(A\colon B\colon C)}$ by substituting $a = A/C$ and avoiding the inversion by carrying the denominator C through to the projective output. All of these functions are summarized in Table 1, where we use an asterisk $(*)$ to denote that they have been modified to work on $E_{(A\colon B\colon C)}$ rather than $E_{(a,b)}$. Conveniently, all of these subroutines are only needed to work entirely in only one of $E_{(A\colon B\colon C)}$ and $E_{(a,b)}$.

During the computations of shared secrets, we found it advantageous to employ the function $\texttt{LADDER\_3\_pt} \colon (x(P), x(Q), x(Q-P), a, m) \mapsto x(P + [m]Q)$, which is precisely the "three point ladder" given by De Feo, Jao and Plût [19, Algorithm 1].

Following [19], we also derived a function to compute $\texttt{xTPL} \colon (x(P), A, C) \mapsto x([3]P)$ on $E_{(A\colon B\colon C)}/\{\pm 1\}$. To our knowledge, our derived tripling formulas are the fastest known x-only Montgomery tripling formulas, so we briefly describe their derivation. We take the tripling to be the concatenation of a doubling and an addition operation, i.e.,

$$\texttt{xTPL}(x(P), a) = \texttt{xADD}\big(\texttt{xDBL}(x(P), a), x(P), x(P), a\big).$$

Let $P = (X : Y : Z)$ with $Z \neq 0$ such that $[3]P \neq \mathcal{O}$, and[8] $x_P = X/Z$. Simplifying the formulas for tripling P obtained from the above concatenation yields

$$x_{[3]P} = x_P \cdot \left(\frac{x_P^4 - 6x_P^2 + 4ax_P + 3}{3x_P^4 + 4ax_P^3 + 6x_P^2 - 1} \right)^2,$$

where $x([3]P) = (X_3 : Z_3)$ and $x_{[3]P} = X_3/Z_3$. Substituting $a = A/C$ and $x_P = X/Z$, we get that $(X_3 \colon Z_3)$ can be computed from $(X \colon Z)$, A and C in either $9\mathbf{M} + 4\mathbf{S} + 12\mathbf{a}$ or $8\mathbf{M} + 5\mathbf{S} + 15\mathbf{a}$[9].

[8] Throughout we use x_P to denote the affine x-coordinate of $P \neq \mathcal{O}$, i.e. $x_P = X/Z$ is a unique normalized value, in contrast with $x(P)$ which denotes a projective point $x(P) = (X \colon Z) \in \mathbb{P}^1$ that can be represented by any X, Z such that $(X \colon Z) \sim (x_P \colon 1)$.

[9] As usual, $\mathbf{M}$, $\mathbf{S}$ and $\mathbf{a}$ represent the costs of field multiplications, squarings, and additions, respectively. We will always count multiplications by curve coefficients as full multiplications, since these coefficients change within an isogeny class and thus we cannot expect any savings by treating them differently to generic elements.

Here we are counting multiplications by A and C as $1\mathbf{M}$ each, and we note that two of the tallied multiplications are multiplications by C, i.e., on a fixed Montgomery curve, both sets of formulas become $2\mathbf{M}$ faster.

Minimizing the number of inversions via curves in $\mathbb{P}^1$. Observe that all of the functions mentioned above on $E_{(a,b)}/\{\pm 1\}$ (resp. $E_{(A\colon B\colon C)}/\{\pm 1\}$) depend entirely on a (resp. A and C) and are independent of b (resp. B). This is because, for a fixed $a = A/C$ and up to isomorphism, there are only two curves found by varying b (resp. B) over K: the curve E and its non-trivial quadratic twist. Indeed, an elliptic curve and its twist are unified under the quotient by $\{\pm 1\}$, i.e., have the same Kummer variety, so it is no surprise that the Kummer arithmetic is independent of the Montgomery b (resp. B) coefficient. Moreover, referring back to (1), we see that the j-invariant is also independent of b (resp. B)[10].

Our implementation profits significantly from these observations, and the choice of Montgomery form provides two advantages in parallel. The first is the well-known Montgomery-style *point* arithmetic that unifies points and their inverses by ignoring the Y coordinate to work with $(X\colon Z) \in \mathbb{P}^1$; the second is new *isogeny* arithmetic that unifies curves and their quadratic twists by ignoring the B coefficient to instead work only with $(A\colon C) \in \mathbb{P}^1$. In this way all point operations and isogeny computations are performed in $\mathbb{P}^1$, meaning that only one inversion is required (at the very end) when generating public keys or computing shared secrets. In the latter case, the inversion is computed during the j-invariant function `j_inv`: $(A, C) \mapsto j(E_{(A\colon B\colon C)})$, while in the former case we use a 4-way simultaneous inversion [39] to normalize all of the components of the public key prior to transmission; see Table 1 for more details on these functions.

Notation and terminology. Strictly speaking, in dropping the b (resp. B) coefficient from our isogeny computations, we will not be deriving isogenies between curves, but maps between their Kummer varieties. Thus, unless otherwise specified, henceforth we use E_a or $E_{(A\colon C)}$ interchangeably to denote the Kummer variety of any K-rational Montgomery curve whose x^2 coefficient is $a = A/C$, or by abuse of notation, to denote any K-rational Montgomery curve with coefficient $a = A/C$. We will continue to use the terminology of, e.g., computing isogenies "between curves", with the understanding that our computations are technically taking place between the associated Kummer varieties. We stress that using Kummer varieties in this way does not affect the security of the scheme (see Remark 1), but has major efficiency benefits.

Projective three isogenies. Let $x(P) = (X_3\colon Z_3) \in \mathbb{P}^1$ be such that P has order 3 in $E_{(A\colon C)}$. Let $E'_{(A'\colon C')} = E_{(A\colon C)}/\langle P\rangle$, $\phi\colon E_{(A\colon C)} \to E'_{(A'\colon C')}$, $Q \in E_a \setminus \ker(\phi)$, and write $x(Q) = (X\colon Z) \in \mathbb{P}^1$ with $x(\phi(Q)) = (X'\colon Z') \in \mathbb{P}^1$. Our goal is to derive two sets of explicit formulas: the first set computes the isogenous curve $E_{(A'\colon C')}$ from $(X_3\colon Z_3)$ and $E_{(A\colon C)}$, while the second set is used to evaluate the corresponding isogeny by computing $(X'\colon Z')$ from the additional input $(X\colon Z)$. The projective version of [19, Equation (17)] gives

$$(A'\colon C') = \big((AX_3Z_3 + 6(Z_3^2 - X_3^2))X_3 \,:\, CZ_3^3\big),$$

which can be computed in $6\mathbf{M}+2\mathbf{S}+5\mathbf{a}$. However, it is possible to do much better by using $Z_3 \neq 0$ and the fact that X_3/Z_3 is a root of the 3-division polynomial $\psi_3(x) = 3x^4 + 4(A/C)x^3 + 6x^2 - 1$ on $E_{(A\colon C)}$. This yields the alternative expression

$$(A'\colon C') = \big((Z_3^4 + 18X_3^2Z_3^2 - 27X_3^4 \,:\, 4X_3Z_3^3\big), \tag{2}$$

which is independent of the coefficients of $E_{(A\colon C)}$ and can be computed in $3\mathbf{M} + 3\mathbf{S} + 8\mathbf{a}$; see the function `get_3_isog` in Table 1. For the evaluation of the isogeny, we modify the map in [19, Equation (17)] to give

$$(X'\colon Z') = \big(X(X_3X - Z_3Z)^2 \,:\, Z(Z_3X - X_3Z)^2\big)\,. \tag{3}$$

This costs $6\mathbf{M} + 2\mathbf{S} + 2\mathbf{a}$; see the function `eval_3_isog` in Table 1.

[10] This is a special case of [51, III.1.4(b)]

Projective four isogenies. We now let $x(P) = (X_4 \colon Z_4) \in \mathbb{P}^1$ be such that P has exact order 4 in $E_{(A\colon C)}$, and leave all other notation and definitions as above. As is discussed in [19, §4.3.2], there are some minor complications in the derivation of 2- and 4-isogenies, either because a direct application of Vélu's formulas [56] for a 2-isogeny do not preserve the Montgomery form, or because repeated application of the 4-isogeny resulting from Vélu's formulas is essentially degenerate. For our purposes, i.e., in the case of 4-isogenies (overall, we found using 4-isogenies to be significantly faster than using 2-isogenies), the latter problem is remedied by application of the simple isomorphism in [19, Equation (15)]. When building the 4^e isogenies as a composition of 4-isogenies, this isomorphism is needed in every 4-isogeny computation except for the very first one, and we derive explicit formulas for both of these cases.

Note that for the very first 4-isogeny $\phi_0 \colon E_{(A\colon C)} \to E_{(A'\colon C')}$ computed in the public key generation phase, the curve $E_{(A\colon C)}$ is that which is specified in the system parameters; and, for the first 4-isogeny in the shared secret computation, $E_{(A\colon C)}$ is the curve that is received as part of a public key sent over the wire. In both cases the curve is normalized so that $A = a$ and $C = 1$. In this case we use [19, Equation (20)] directly, which gives

$$(A' \colon C') = (\, 2(a+6) \,:\, a-2\,)\,, \tag{4}$$

and projectivize the composition of [19, Equations (19) and (21)] to give

$$(X' \colon Z') = \big(\, (X+Z)^2(aXZ + X^2 + Z^2) \,:\, (2-a)XZ(X-Z)^2 \big)\,. \tag{5}$$

This costs $4\mathbf{M} + 2\mathbf{S} + 9\mathbf{a}$; see the function `first_4_isog` in Table 1.

For the general 4-isogeny, we projectivized the composition of the above isogeny with the isomorphism in [19, Equation (15)], making some modifications as follows. We made use of the `xDBL` function to parameterize the point of order 2 in [19, Equation (15)] in terms of the point $(X_4 \colon Z_4)$ of order 4. For the isogeny evaluation function, we again found it advantageous to simplify under the applicable component of the 4-division polynomial $\psi_4(x,y) = 4y(x-1)(x+1)\hat{\psi_4}(x)$, which is $\hat{\psi_4}(x) = x^4 + 2(A/C)x^3 + 6x^2 + 2(A/C)x + 1$ and which vanishes at X_4/Z_4. For the computation of the isogenous curve, we get

$$(A' \colon C') = \big(2(2X_4^4 - Z_4^4) \,:\, Z_4^4 \big), \tag{6}$$

and for the evaluation of the isogeny, we get

$$\begin{aligned}(X' \colon Z') = \big(\, & X \left(2X_4Z_4Z - X(X_4^2 + Z_4^2)\right)(X_4X - Z_4Z)^2 \,: \\ & Z\left(2X_4Z_4X - Z(X_4^2+Z_4^2)\right)(Z_4X - X_4Z)^2 \big).\end{aligned} \tag{7}$$

Since each 4-isogeny is evaluated at multiple points, during the computation of the isogenous curve in (6), we also compute and store five values that can be (re)used in (7): $\mathbf{c} = [\ X_4^2 + Z_4^2,\ X_4^2 - Z_4^2,\ 2X_4Z_4,\ X_4^4,\ Z_4^4\ ]$.

The computation of (6) and of the five values in $\mathbf{c}$ above costs $5\mathbf{S} + 7\mathbf{a}$, and on input of $\mathbf{c}$ and $Q = (X \colon Z)$, the computation of (7) costs $9\mathbf{M} + 1\mathbf{S} + 6\mathbf{a}$; see the functions `get_4_isog` and `eval_4_isog` in Table 1.

Summary of subroutines. All of the point and isogeny operations are summarized in Table 1. We note that the input $\mathbf{c} \in K^5$ into the `eval_4_isog` function is the same tuple of constants output from `get_4_isog`, as described above. We also point out that all of the operation counts in Table 1 are counted as being field operations in K, but that K is not necessarily the same field for all of these functions. In particular, we start the next section by fixing K, but as we will see in §6, the `secret_pt` function (and the `LADDER` functions it calls) takes place in the quadratic subfield of K in our implementation.

Remark 1. We conclude this section by reiterating that the shared secrets derived by working with projective Kummer points and projective Kummer variety coefficients are exactly the same

function /	**input(s)**	**output(s)**	**operations**			
from	**type(s)**	**type(s)**	**M**	**S**	**a**	**I**
`j_inv`	(A, C)	$j(E)$	3	4	8	1
Eq. (1)	$K \times K$	K				
`xDBLADD`	$(x(P), x(Q), x(Q-P), \hat{a})$	$(x([2]P), x(Q+P))$	6	4	8	-
[39]	$\mathbb{P}^1 \times \mathbb{P}^1 \times \mathbb{P}^1 \times K$	$\mathbb{P}^1 \times \mathbb{P}^1$				
`xADD`	$(x(P), x(Q), x(Q-P))$	$x(Q+P)$	3	2	6	-
[39]	$\mathbb{P}^1 \times \mathbb{P}^1 \times \mathbb{P}^1$	$\mathbb{P}^1$				
`xDBL`	$\big(x(P), \hat{A}, \hat{C}\big)$	$x([2]P)$	4	2	4	-
[39]*	$\mathbb{P}^1 \times K \times K$	$\mathbb{P}^1$				
`xDBLe`	$(x(P), A, C, e)$	$x([2^e]P)$	4e	2e	$4e$	-
[39]*	$\mathbb{P}^1 \times K \times K \times \mathbb{N}$	$\mathbb{P}^1$			$+3$	
`LADDER`	$\big(x(P), a, m\big)$	$x([m]P)$	$5n$	$4n$	$9n$	-
[39]	$\mathbb{P}^1 \times K \times \mathbb{N}$	$\mathbb{P}^1$				
`LADDER_3_pt`	$(x(P), x(Q), x(Q-P), a, m)$	$x(P+[m]Q)$	$9n$	$6n$	$14n$	-
[19]	$\mathbb{P}^1 \times \mathbb{P}^1 \times \mathbb{P}^1 \times K \times \mathbb{N}$	$\mathbb{P}^1$			$+3$	
`xTPL`	$\big(x(P), \hat{A}, C\big)$	$x([3]P)$	8	5	15	-
[19]*	$\mathbb{P}^1 \times K \times K$	$\mathbb{P}^1$				
`xTPLe`	$(x(P), A, C, e)$	$x([3^e]P)$	8e	5e	$15e$	-
[19]*	$\mathbb{P}^1 \times K \times K \times \mathbb{N}$	$\mathbb{P}^1$				
`get_3_isog`	$x(P)$	(A', C')	3	3	8	-
[19]* and (2)	$\mathbb{P}^1$	$K \times K$				
`eval_3_isog`	$(x(P), x(Q))$	$x(\phi(Q))$	6	2	2	-
[19]*	$\mathbb{P}^1 \times \mathbb{P}^1$	$\mathbb{P}^1$				
`first_4_isog`	$(x(Q), a)$	$(x(\phi_0(Q)), A', C')$	4	2	9	-
[19]* and (4)-(5)	$\mathbb{P}^1 \times K$	$\mathbb{P}^1 \times K \times K$				
`get_4_isog`	$x(P)$	$(A', C', \mathbf{c})$	-	5	7	-
[19]* and (6)	$\mathbb{P}^1$	$K \times K \times K^5$				
`eval_4_isog`	$(\mathbf{c}, x(Q))$	$x(\phi(Q))$	9	1	6	-
[19]* and (7)	$K^5 \times \mathbb{P}^1$	$\mathbb{P}^1$				
`secret_pt`	$(P, Q = \tau(P), m)$	$x(P+[m]Q)$	$5n$	$4n$	$9n$	-
see §6	$E \times E \times \mathbb{N}$	$\mathbb{P}^1$	$+16$	$+3$	$+11$	
`distort_and_diff`	x_P	$x(\tau(P) - P)$	-	1	2	-
see §6	K	$\mathbb{P}^1$				
`inv_4_way`	(z_1, z_2, z_3, z_4)	$(z_1^{-1}, z_2^{-1}, z_3^{-1}, z_4^{-1})$	9	-	-	1
[39]	K^4	K^4				

Table 1. Summary of the subroutines used in our SIDH implementation. Here the points P and Q are on the curve $E_{(a,b)} = E_{(A\colon B\colon C)}$, and $E' = E_{(A'\colon B'\colon C')}$ is used to denote the isogenous curve $E' = E/\langle P \rangle$ with $\phi\colon E \to E'$ the corresponding isogeny. Asterisks (*) are used in the first column when functions have been modified according to our use of the curve model $E_{(A\colon B\colon C)}$, rather than $E_{(a,b)}$. We use $n = \log_2 m - 1$ to count operations, and note some minor changes in input constants: $4\hat{a} = a + 2$ in `xDBLADD`, $4\frac{\hat{A}}{\hat{C}} = \frac{A}{C} + 2$ in `xDBL`, $\hat{A} = 4A$ in `xTPL`.

as the shared secrets that would result from instead working in, e.g., affine space using Vélu's formulas for the full short Weierstrass model [56]. Quite trivially, since the derived j-invariants only depend on the $\bar{K}$-isomorphism class of a curve [51, III.1.4(b)], using isomorphic curve models or working with the Kummer variety that corresponds to both quadratic twists cannot change the result. As such, the improvements in this section can be viewed as implementation choices that aid efficiency and compactness but do not necessarily hamper interoperability, e.g., if the other party chooses to compute their keys in a different manner. In the next section we will define our starting curve E_0, which in particular is both a Montgomery and short Weierstrass curve. The script `Kummer_Weierstrass_equivalence.mag` is included in [16] to illustrate this point. In particular, it performs each round of the SIDH key exchange both using our projective Kummer variety arithmetic and using Magma's [10] built-in `IsogenyFromKernel` function, in order to show that the resulting curves in the public keys are isomorphic and that the shared secrets are equivalent.

4 Parameters and implementation choices

In this section we fix the underlying starting curve and discuss several implementation choices that were made in the development of our library.

Prime field and isogeny class. From here on, the field K is fixed as $K := \mathbb{F}_{p^2}$, where

$$p := 2^{372} \cdot 3^{239} - 1\,,$$

and $\mathbb{F}_{p^2} = \mathbb{F}_p(i)$ for $i^2 = -1$. In terms of the notation from Section 2, this means that $\ell_A = 2$, $\ell_B = 3$, $e_A = 372$, $e_B = 239$ and $f = 1$. We searched for primes of the form $2^{e_A}3^{e_B}f - 1$ with a bit length close to (but no larger than) 768, aiming to strike a balance $\ell_A^{e_A} \approx \ell_B^{e_B}$ to ensure that one side of the key exchange is not appreciably easier to attack than the other (more on this below), and to balance the computational costs for Alice and Bob. We originally searched with no restriction on the cofactor f, but did not find an example of another prime that would perform as fast as ours and where the overall security was increased enough to warrant $f \neq 1$. Given the best known classical and quantum attack complexities (see Section 1), choosing a prime close to 768 bits aims to reach a claim of 192 bits of classical security and 128 bits of quantum security. Following [19, §5.1], and since $\ell_A^{e_A} < \ell_B^{e_B}$ and $\ell_A = 2$, we ignore any constants hidden by the big-O notation to err on the conservative side and claim that the above curve offers $e_A/2 = 186$ bits of classical security and $e_A/3 = 124$ bits of quantum security. The arithmetic advantages of this prime choice are detailed in Section 5.

Our implementation works in the isogeny class of elliptic curves over $\mathbb{F}_{p^2}$ that contains the supersingular Montgomery curve

$$E_0/\mathbb{F}_{p^2} \colon y^2 = x^3 + x.$$

Every curve in this isogeny class has $(p+1)^2 = (2^{372} \cdot 3^{239})^2$ points and is also supersingular [51, Exercise 5.4 & 5.10(a)]. The curve E_0 is the public parameter that is the starting point for the key exchange protocol.

Remark 2 (Starting on a subfield curve). In [8], Biasse, Jao and Sankar give a quantum algorithm that solves the more general (i.e., unknown degree) supersingular isogeny problem. To compute an isogeny between $E_1/\mathbb{F}_{p^2}$ and $E_2/\mathbb{F}_{p^2}$, their algorithm finds an isogeny path from $E_1/\mathbb{F}_{p^2}$ to a subfield curve $E_1'/\mathbb{F}_p$ and an isogeny path from $E_2/\mathbb{F}_{p^2}$ to a subfield curve $E_2'/\mathbb{F}_p$, before connecting the paths by finding an isogeny between $E_1'/\mathbb{F}_p$ and $E_2'/\mathbb{F}_p$. In both cases the search for an isogenous subfield curve has complexity $\tilde{O}(p^{1/4})$, and the step that finds a path between E_1' and E_2' takes subexponential time. This yields an overall attack complexity of $\tilde{O}(p^{1/4})$, and is therefore not competitive with the quantum claw attack mentioned in Section 1. Nevertheless, Biasse, Jao and Sankar [8, §6] suggest that the nature of their attack (and the possibility of an improved version) could mean that choosing a subfield curve as the starting point is less secure than starting from a curve defined over $\mathbb{F}_{p^2}$; indeed, if E_1 is already defined over $\mathbb{F}_p$ then the $\tilde{O}(p^{1/4})$ step need only be performed once instead of twice.

In our implementation we still choose to start on a subfield curve for one main reason. To our knowledge, there is currently no known way to randomly sample from the isogeny class directly, i.e., without (some party) knowing the isogeny path to a subfield curve. Indeed, over the fields of interest in SIDH, Bröker's construction [11] outputs a subfield curve $E/\mathbb{F}_p$, and De Feo, Jao and Plût [19] say that one can find a different starting curve that is minimally defined over $\mathbb{F}_{p^2}$ by means of a random walk (starting from E) on the isogeny graph. While it is possible that there are already cryptographic techniques available that allow two or more mutually distrusting parties to arrive at a starting curve for which no single party knows the route to a subfield curve, until a more satisfactory solution is developed (e.g., a way of hashing directly into the isogeny class) it seems most prudent to *lay ones cards on the table* and start with E_0 as above. Moreover, we reiterate that such a special choice of starting curve makes no known difference to the complexity of the best known attacks, and as we are about to show, can give rise to more compact public parameters and more efficient key generation algorithms.

The base-field and trace-zero torsion subgroups. A valuable technique that was introduced by Verheul [57] and that has played a key role in the implementation of symmetric pairings on supersingular elliptic curves [48], is that of using a distortion map. Verheul showed that every supersingular elliptic curve has a distortion map [57]. For a prime power $\ell^e \mid \#E_0(\mathbb{F}_p)$, such a map connects the cyclic torsion subgroup $E_0(\mathbb{F}_p)[\ell^e]$ defined over the base field $\mathbb{F}_p$ with the trace-zero subgroup of $E_0(\mathbb{F}_{p^2})[\ell^e]$.

In the pairing context, this technique essentially enables one to work mostly with points defined over the base field instead of the quadratic extension. The distortion map we use for E_0 is given by the endomorphism $\tau : E_0(\mathbb{F}_{p^2}) \to E_0(\mathbb{F}_{p^2}),\ (x, y) \mapsto (-x, iy)$.

An ℓ^e torsion point $P \in E_0(\mathbb{F}_p)$ is mapped to an ℓ^e-torsion point $\tau(P) \in E_0(\mathbb{F}_{p^2})$ and the Weil pairing $e_{\ell^e}(P, \tau(P)) \neq 1$ is non-trivial. It is easy to see that the trace of the image point is zero, namely $\mathrm{Tr}(\tau(P)) = \tau(P) + \pi_p(\tau(P)) = \mathcal{O}$, where π_p is the p-power Frobenius endomorphism on E_0. An advantage of using the trace-zero subgroup is that its points can be represented by two $\mathbb{F}_p$-elements only and are therefore half the size of a general curve point defined over $\mathbb{F}_{p^2}$.

Choosing generator points for torsion subgroups. We apply a similar idea in that we fix the public $\ell_A^{e_A}$-torsion points P_A, Q_A and $\ell_B^{e_B}$-torsion points P_B, Q_B as generators of the (respective) base field and trace-zero subgroups, chosen as follows. Let $P_A \in E_0(\mathbb{F}_p)[2^{372}]$ be the point given as $[3^{239}](z, \sqrt{z^3 + z})$, where z is the smallest positive integer such that $\sqrt{z^3 + z} \in \mathbb{F}_p$ and P_A has order 2^{372}. The point P_B is selected in the same way with order and cofactor swapped. We then take $Q_A = \tau(P_A)$ and $Q_B = \tau(P_B)$, which produces the following generators:

$$P_A = [3^{239}](11, \sqrt{11^3 + 11}), \qquad Q_A = \tau(P_A),$$
$$P_B = [2^{372}](6, \sqrt{6^3 + 6}), \qquad Q_B = \tau(P_B).$$

In addition to the base field representations mentioned above, the simple relationship between the coordinates of Q_A and P_A and the coordinates of Q_B and P_B helps to further compactify the public parameters; see Section 6. However, choosing $\{P_A, Q_A\}$ and $\{P_B, Q_B\}$ as the bases for generating isogeny kernels from the base-field and trace-zero torsion subgroups can have caveats. For example, in the case $\ell = \ell_A = 2$, one obtains the following lemma.

Lemma 1. *Let $E : y^2 = x^3 + x$ be a supersingular elliptic curve defined over $\mathbb{F}_p$, $p > 3$, $p \equiv 3 \pmod 4$, such that $\#E(\mathbb{F}_p) = 2^e \cdot N$ with N odd. Let $\mathbb{F}_{p^2} = \mathbb{F}_p(i)$, $i^2 = -1$, and let $E[\ell^e] \subseteq E(\mathbb{F}_{p^2})$. Let $P \in E(\mathbb{F}_p)[2^e]$ be any point of order 2^e and let $Q \in E(\mathbb{F}_{p^2})[2^e]$ be any point of order 2^e with $\mathrm{Tr}(Q) = Q + \pi_p(Q) = \mathcal{O}$. Then the order of $P + Q$ equals 2^{e-1}.*

Proof. The distortion map $\tau : E(\mathbb{F}_{p^2}) \to E(\mathbb{F}_{p^2}),\ (x, y) \mapsto (-x, iy)$ is a group isomorphism from the cyclic group $\langle P \rangle = E(\mathbb{F}_p)[2^e]$ to the trace-zero subgroup of $E(\mathbb{F}_{p^2})[2^e]$. In particular, this means that $Q = \tau(Q')$ for a point $Q' \in E(\mathbb{F}_p)[2^e]$. This means that $Q' = [\lambda]P$ for $\lambda \in \mathbb{Z}$, $1 \leq \lambda \leq 2^e - 1$ and $\lambda = 1 + 2\lambda'$ must be odd because Q has full order. We thus see that

$$P + Q = P + \tau(Q') = P + \tau([\lambda]P) = P + [\lambda]\tau(P) = (P + \tau(P)) + [2\lambda']\tau(P).$$

The point $[2^{e-1}]P$ has order 2 and is defined over $\mathbb{F}_p$. The only such point is $(0, 0)$, which means that $[2^{e-1}]P = (0, 0)$. It follows that

$$[2^{e-1}](P + \tau(P)) = [2^{e-1}]P + \tau([2^{e-1}]P) = (0, 0) + \tau((0, 0)) = \mathcal{O}.$$

This proves that

$$[2^{e-1}](P + Q) = [2^{e-1}](P + \tau(P) + [2\lambda']\tau(P)) = [2^{e-1}](P + \tau(P)) + [\lambda']([2^e]\tau(P)) = \mathcal{O}.$$

To see that $P+Q$ has the exact order 2^{e-1}, assume that it has smaller order, i.e. $[2^{e-2}](P+Q) = \mathcal{O}$. Then $[2^{e-2}]P = -[2^{e-2}]Q$. The point $[2^{e-2}]P$ has order 4 and is defined over $\mathbb{F}_p$. Such a point does not have trace zero, and thus cannot be equal to a non-zero multiple of the point Q, which lies in the trace-zero subgroup. □

In particular, Lemma 1 proves that any point of the form $P + [m]Q$ for odd m has order less than 2^e. Also note that if m is even, then the order of $P + [m]Q$ is 2^e because $[2^{e-1}](P + [m]Q) = [2^{e-1}]P \neq \mathcal{O}$. Furthermore, this means that the points P and Q do not generate the full 2^e-torsion subgroup, and strictly speaking, the two points are not independent[11].

Recalling that the best attacks against SSDDH assume that the isogeny degrees are known to be maximal, we remark that the implementation is also much easier if this is always the case; among other things, it fixes the optimal strategies for isogeny computations and simplifies the techniques for obtaining constant-time code. In the following two paragraphs we show how Alice and Bob can choose their secret scalars to guarantee that the degrees of their isogenies are maximal, i.e., $\ell_A^{e_A}$ and $\ell_B^{e_B}$ respectively.

Sampling full order 2-torsion points. To sample a 2-torsion point R_A of full order, we sample a uniform random integer $m' \in \{1, 2, \ldots, 2^{e_A-1} - 1 = 2^{371} - 1\}$ and set $R_A = P_A + [2m']Q_A$; R_A is guaranteed to have order 2^{e_A} by the above discussion. Because two distinct choices for m' lead to two distinct cyclic subgroups generated by the corresponding R_A, one can reach $2^{e_A-1} - 1 = 2^{371} - 1$ distinct subgroups and thus isogenies with this sampling procedure. We have seen in Section 2 that there are $3 \cdot 2^{e_A-1}$ distinct full order subgroups in $E_0[2^{e_A}]$, and thus our sampling procedure only reaches about one third of those.

We have restricted to this case since it allows us to obtain a simpler implementation than when all of the other cases (such as $[2m']P_A + Q_A$) are covered in order to span the full set of possible isogenies. It is an interesting and open problem to determine whether choices such as this have implications on the security of the protocol; see Remark 5 for further discussion.

Sampling full order 3-torsion points. To sample a 3-torsion point R_B of full order, we sample a uniform random integer $m' \in \{1, 2, \ldots, 3^{e_B-1} - 1 = 3^{238} - 1\}$ and set $R_B = P_B + [3m']Q_B$. Since $[3^{e_B-1}]R_B = [3^{e_B-1}]P_B \neq \mathcal{O}$, R_B is guaranteed to have order 3^{e_B}. In this way, we reach $3^{238} - 1$ of the possible subgroups and corresponding isogenies. Since there are $4 \cdot 3^{e_B-1}$ such subgroups in $E_0[3^{e_B}]$, we sample from about one quarter of those.

Again, for simplicity of the implementation, we chose to restrict to the above subclass of isogenies and to leave the security implications of that choice as an open problem for future work; see Remark 5.

[11] Whenever we use the term independent for the points P and Q in what follows, we mean that the Weil pairing evaluated at P and Q is non-trivial.

Remark 3. Choosing to define $Q_A = \tau(P_A)$ and $Q_B = \tau(P_B)$ is independent of whether or not the base field and/or trace-zero subgroups are used in the definition of the public parameters, and in any case it offers the advantage of more compact public parameters mentioned above. In general, whether the generators are related by the distortion map or chosen independently (by multiplying random points by the corresponding cofactors as suggested in [19]), it is both possible that the resulting pairs of points do generate the full ℓ^e-torsion, and possible that they do not. In both cases, the order of the Weil pairing allows us to check the proportion of the torsion generated by P_A and Q_A (or P_B and Q_B); we note that this check is once-and-for-all and can be done offline. Once more, the security implications of linking the generators with a distortion map are unknown and left open.

Strategies for isogeny computation and evaluation. For computing and evaluating $\ell_A^{e_A}$- and $\ell_B^{e_B}$-isogenies, we closely follow the methodology described in [19, §4.2.]. As already described in Section 2, such isogenies are composed of e_A isogenies of degree ℓ_A and e_B isogenies of degree ℓ_B, respectively. Figure 2 in [19] illustrates this computation with the help of a directed acyclic graph. In order to be able to evaluate the desired isogeny, one needs to compute all points that are represented by the final vertices, i.e., the leaves in the graph.

As described earlier in Section 2, using the multiplication-based or isogeny-based methods to traverse this graph yields a simple but costly algorithm. De Feo, Jao, and Plût [19, §4.2.2] provide a discussion of how to obtain an optimal algorithm. They formally define the notion of a *strategy* for evaluating ϕ along a directed acyclic graph and show how to find an optimal strategy depending on the relative costs of scalar multiplication-by-ℓ and ℓ-isogeny evaluation. The overall cost measure is the total cost for computing and evaluating the isogeny. The key insight is that an optimal strategy must be composed of two optimal sub-strategies, which yields a dynamic programming approach to optimizing the overall strategy by working our way up from small ℓ-power isogenies to higher powers.

A strategy corresponds to a binary tree topology and hence a strategy for a tree with a given number of leaves can be decomposed into two sub-strategies on strictly smaller leaf sets. The overall cost for the strategy is the sum of the sub-strategy costs plus the cost for moving down the tree along the edges to the roots of the sub-strategies. Given the costs of all optimal sub-strategies, one can select the optimal strategy by going through all possibilities of combining two of them. We refrain from giving more details for space reasons and refer to [19, §4.2.2].

A strategy can be stored in a simple list L of integers of length equal to the total number of leaves. The i-th entry $L[i]$ in the list characterizes the sub-strategy on a graph with i leaves. In our setting, we chose to use the number of scalar-multiplication steps along the left edge of the graph that are needed in a strategy on i leaves to reach the root of the next sub-strategy. This number uniquely determines the splitting of the strategy into its sub-strategies. The sub-strategies are then defined by the respective sub-sequences. At the same time the list L determines an algorithm to traverse the graph (as shown for example in Algorithm 3).

The strategies for 4-isogeny and 3-isogeny computation that we use in our implementation are given below; they were derived on input of the following cost ratios: the ratio between the latencies of a 4-isogeny evaluation (`eval_4_isog`) and the multiplication-by-4 map (two applications of `xDBL`) was $22.8/(2\cdot 12.9)$, and the ratio between the latencies of a 3-isogeny evaluation (`eval_3_isog`) and the multiplication-by-3 map (`xTPL`) was $17.0/27.8$. These ratios were obtained on an Intel Haswell machine running Ubuntu 14.04 LTS after averaging 10^3 executions of each of the functions and rounding the values to the nearest 10^2 clock cycles. Alice's optimal strategy for the 4-isogeny computation is described by L_A and Bob's optimal strategy for the 3-isogeny is described by L_B. The script `optimalstrategies.mag` in [16] can be modified to generate such strategies in other situations (e.g., over other primes and for different cost ratios).

$$\begin{aligned} L_A = [&0, 1, 1, 2, 2, 2, 3, 4, 4, 4, 4, 5, 5, 6, 7, 8, 8, 8, 8, 8, 9, 10, 9, 12, 11, 11, 12, 12, 13, 14, 15, 16, 16, 16, 16, 16, 17, 17, 17,\\ &17, 17, 19, 19, 17, 18, 19, 20, 21, 22, 21, 23, 22, 24, 24, 25, 25, 27, 27, 27, 28, 30, 30, 31, 32, 32, 33, 33, 33, 33, 32, 33,\\ &33, 33, 33, 33, 33, 33, 33, 36, 34, 35, 34, 35, 38, 37, 38, 38, 39, 38, 41, 39, 43, 38, 41, 42, 43, 43, 40, 41, 42, 43, 44, 45,\\ &46, 47, 48, 49, 50, 48, 49, 53, 51, 51, 51, 53, 55, 56, 55, 56, 58, 58, 58, 59, 61, 61, 63, 63, 64, 64, 64, 65, 65, 65, 64, 64,\\ &65, 65, 65, 66, 67, 65, 66, 65, 68, 66, 65, 66, 65, 66, 67, 65, 66, 67, 68, 69, 70, 71, 72, 71, 72, 71, 76, 71, 76, 72, 71, 76,\\ &71, 73, 72, 76, 76, 73, 73, 72, 76, 76, 75, 76, 76, 75, 81, 81, 83, 81]. \end{aligned}$$

$$\begin{aligned} L_B = [&0, 1, 1, 2, 2, 2, 3, 3, 4, 4, 4, 5, 5, 5, 6, 7, 8, 8, 8, 8, 9, 9, 9, 9, 9, 12, 12, 12, 12, 12, 12, 12, 13, 14, 14, 15, 16, 16, 16, 16,\\ &17, 16, 19, 17, 19, 19, 19, 20, 21, 22, 22, 22, 22, 22, 22, 22, 24, 22, 22, 24, 24, 26, 27, 27, 28, 28, 28, 30, 28, 28, 28, 29,\\ &28, 28, 28, 29, 29, 30, 33, 33, 33, 33, 34, 35, 37, 37, 37, 38, 38, 38, 37, 38, 38, 38, 38, 38, 39, 38, 44, 43, 44, 39, 40, 41,\\ &43, 43, 43, 45, 46, 46, 46, 47, 48, 48, 49, 49, 50, 51, 51, 49, 49, 50, 51, 50, 51, 50, 50, 51, 50, 51, 51, 51, 53, 55, 55, 55,\\ &56, 56, 56, 56, 56, 57, 58, 61, 61, 61, 63, 63, 63, 64, 65, 66, 65, 66, 66, 66, 65, 66, 66, 66, 66, 66, 68, 71, 66, 66, 68, 67,\\ &71, 66, 66, 68, 67, 71, 66, 66, 68, 68, 71, 70, 70, 72, 72, 76, 75, 75, 78, 78, 78, 80, 80, 80, 80, 81, 81, 81, 82, 83, 84, 85,\\ &86, 86, 86, 86, 86, 86, 88, 86, 90, 86, 92, 87, 86, 89, 86, 92, 87, 86, 87, 86, 91, 89, 89, 90, 90, 92, 92, 92, 93, 93, 93, 95,\\ &95, 95, 95, 95, 95, 95, 95] \end{aligned}$$

5 Field arithmetic

In this section, we describe the advantages of the chosen prime and optimizations to speed up the modular reduction inside SIDH, which were inspired by similar work on so-called Montgomery-friendly primes (e.g., see [33, 25]). We remark that similar ideas can be easily applied to selecting primes and implementing their modular arithmetic at different security levels.

In our case, arithmetic is performed modulo the prime $p = 2^{372} \cdot 3^{239} - 1$. As described in Section 4, choosing an SIDH prime such that $\ell_A^{e_A} \approx \ell_B^{e_B}$ ensures a certain security strength across the whole key exchange scheme. Additionally, some implementations benefit from having a prime with a bit length slightly smaller than a multiple of a word size. Since 768 is the next multiple of 32 and 64 above the bit length of our prime, and $\log_2 p = 751 = 768 - 17$, the *extra room* available at the word boundaries enables the efficient use of other optimization techniques such as carry-handling elimination, and eases the efficient use of vector instructions. Working on a field of size slightly smaller than 2^{768} enables us to, e.g., use 12×64-bit limbs to represent field elements, whereas a prime slightly larger than 2^{768}, such as $p_{768} = 2^{387} \cdot 3^{242} - 1$ from [2], requires 13×64-bit limbs; the latter choice brings a relatively small increase in security at the expense of a significant increase in the cost of the modular arithmetic.

Since we work over $\mathbb{F}_{p^2}$, where $\mathbb{F}_{p^2} = \mathbb{F}_p(i)$ for $i^2 = -1$, we can leverage the extensive research done on the efficient implementation of such quadratic extension fields. In the context of pairings, high-speed implementations have exploited the combination of Karatsuba multiplication, lazy reduction, and carry-handling elimination; e.g., these techniques have been combined in optimized implementations on the curve BN254 [1]. Here we can follow a similar strategy since our field definition and underlying prime share several common traits with BN254, e.g., our prime being slightly smaller than a multiple of the word size enables the computation of several additions without carry-outs in the most significant word.

Efficient modular reduction. The cost of modular arithmetic (and, in particular, of modular multiplication) dominates the cost of the isogeny-based key exchange, so its efficient implementation is crucial for achieving high performance. At first glance, it would seem that SIDH primes prompt the use of generic Montgomery [38] or Barrett [5] reduction algorithms, which are relatively expensive in comparison with the efficient reduction of certain primes with special form (e.g., pseudo-Mersenne primes). For example, Azarderakhsh, Fishbein and Jao [2] use a generic Barrett reduction for computing the modular multiplication in their SIDH implementation. However, we note that primes of this form *do* have a special shape that is amenable to faster modular reduction. Consider the case of the well-known Montgomery reduction [38]: letting $R = 2^{768}$ and $p' = -p^{-1} \bmod R$, then one can compute the Montgomery residue $c = aR^{-1} \bmod p$ for an input $a < pR$, by using

$$c = (a + (ap' \bmod 2^{768}) \cdot p)/2^{768},$$

which costs approximately $s^2 + s$ multiplications for a $2s$-limb value a. For $p = 2^{372} \cdot 3^{239} - 1$, however, this computation simplifies to

$$\begin{aligned} c &= \left(a + (ap' \bmod 2^{768}) \cdot 2^{372} \cdot 3^{239} - (ap' \bmod 2^{768})\right)/2^{768} \\ &= a/2^{768} + \left((ap' \bmod 2^{768}) \cdot 3^{239}\right)/2^{396}. \end{aligned}$$

Moreover, $p' = -p^{-1} \bmod 2^{768}$ also exhibits a special form which reduces the cost of computing $ap' \bmod 2^{768}$ (e.g., $p' - 1$ contains five 64-bit limbs or eleven 32-bit limbs of value 0). In total, the cost of computing c in this case is $s(s - \lfloor 372/w \rfloor)$ multiplications for a word-size w. For example, if $w = 64$ (i.e., $s = 12$), the theoretical speedup for the simplified modular reduction is about 1.85x when applying these optimizations.

It is straightforward to extend the above optimizations to the different Montgomery reduction variants that exist in the literature. For our implementation, we adapted the Comba-based Montgomery reduction algorithm from [47]. Although merged multiplication/reduction algorithms, such as the coarsely integrated operand scanning (CIOS) Montgomery multiplication [31], offer performance advantages in certain scenarios, we prefer an implementation variant that consists of separate routines for integer multiplication and modular reduction. This approach enables the use of lazy reduction for the $\mathbb{F}_{p^2}$ arithmetic and allows easy-to-implement improvements in the integer multiplication, e.g., by using Karatsuba.

Algorithm 1 is based on the Montgomery reduction algorithm in product scanning form (a.k.a. Comba) presented in [47]. It has been especially tailored for efficient computation modulo the prime $p = 2^{372} \cdot 3^{239} - 1$ following the optimizations discussed above. As usual, given a radix-2^r field element representation using s limbs, the algorithm receives as input an operand $a < 2^{rs}p$ (e.g., the integer product of two Montgomery residues) and outputs the Montgomery residue $c = a \cdot 2^{-rs} \bmod p$. Here c is typically computed as $(a + (ap' \bmod 2^r) \cdot p)/2^r$ (s times) in a Comba-like fashion, where $p' = -p^{-1} \bmod 2^r$. However, as mentioned above, this expression simplifies to $(a + (a \bmod 2^r) \cdot \hat{p})/2^r$ where $\hat{p} = p + 1 = 2^{372} \cdot 3^{239}$, since $p' = 1$ for our prime. In addition, Algorithm 1 eliminates several multiplications due to the fact that the $\lfloor e_A/r \rfloor$ least significant limbs in $\hat{p}$ have value 0.

Since our scheme forces the availability of extra room in the radix-2^r representation (which is made possible by having the additional condition that $p < 2^{rs-2}$), there is no overflow in the most significant word during the computation of c in Algorithm 1 (i.e., its intermediate value can be held on exactly s r-bit registers). Moreover, if field elements are represented as elements in $[0, 2p - 1]$ (instead of the typical range $[0, p - 1]$), the output of Algorithm 1 remains bounded without the need of the conditional subtraction in Steps 26-28 [58].

Although typical values for r would be $w = 32$ or 64 to match w-bit architectures, some redundant representations might benefit from the use of $r < w$ in order to avoid additions with carries or to facilitate the efficient use of vector instructions. To this end, the chosen prime is very flexible and supports different efficient alternatives; for example, it supports the use of a 58-bit representation with $s = 13$ limbs when using 64-bit multipliers or the use of a 26-bit representation with $s = 29$ limbs when using 32-bit multipliers.

In our 64-bit implementation, we opted for a generic radix-2^{64} representation using $s = 12$ limbs, in which case the Montgomery constant is $2^{rs} = 2^{768}$. In this case, given that the initial and final loop iterations can be simplified in an unrolled implementation of Algorithm 1, the cost of the modular reduction is 83 multiplication instructions. This result almost halves the number of multiplication instructions compared to the typical Montgomery reduction, which requires $12^2 + 12 = 156$ multiplication instructions (per reduction).

Inversions. SIDH-based key generation and shared secret computations each require one modular inversion at the very end of their executions. These inversions can be implemented using Mont-

Algorithm 1 Optimized Comba-based Montgomery reduction for the prime $p = 2^{372} \cdot 3^{239} - 1$.

Input: The prime $p = 2^{e_A} \cdot 3^{e_B} - 1$; the value $\hat{p} = p + 1$ containing $z = \lfloor e_A/r \rfloor$ 0-value terms in its r-bit representation, where $e_A = 372$, $e_B = 239$ and 2^r is the radix; the Montgomery constant 2^{rs} such that $2^{r(s-1)} \leq p < 2^{rs-1}$; and, the operand $a = (a_{2s-1}, ..., a_1, a_0)$ with $a < 2^{rs}p$ and $s = \lceil \log_2 p/r \rceil$.
Output: The Montgomery residue $c = a \cdot 2^{-rs} \bmod p$.

```
$(t, u, v) = 0$
for $i = 0$ to $s - 1$ do
    for $j = 0$ to $i - 1$ do
        if $j < i - z + 1$ then
            $(t, u, v) = c_j \times \hat{p}_{i-j} + (t, u, v)$
        end if
    end for
    $(t, u, v) = (t, u, v) + a_i$
    $c_i = v$
    $v = u,\ u = t,\ t = 0$
end for
for $i = s$ to $2s - 2$ do
    if $z > 0$ then
        $z = z - 1$
    end if
    for $j = i - s + 1$ to $s - 1$ do
        if $j < s - z$ then
            $(t, u, v) = c_j \times \hat{p}_{i-j} + (t, u, v)$
        end if
    end for
    $(t, u, v) = (t, u, v) + a_i$
    $c_{i-s} = v$
    $v = u,\ u = t,\ t = 0$
end for
$c_{s-1} = v + a_{2s-1}$
if $c \geq p$ then
    $c = c - p$
end if
return $c$
```

gomery inversion based on, e.g., the binary GCD algorithm. However, this method does not run in constant time by default, and therefore requires additional countermeasures to protect it against timing attacks (e.g., the application of input randomization). Since inversion is used scarcely in our software, we instead opted for the use of Fermat's little theorem, which inverts the field element a via the exponentiation $a^{p-2} \bmod p$ that uses a fixed addition chain. Our experiments showed that the cost of this exponentiation is around 9 times slower than (an average run of) the GCD-based method, however even the more expensive inversion only contributes to less than 1% of the overall latency of each round of the protocol. Thus, our choice to compute each isolated inversion via a fixed exponentiation protects the implementation without impacting the performance in any meaningful way, and avoids the need for any additional randomness.

6 SIDH implementation summary

In this section we pull together all of the main ingredients from Sections 2-5 to give a brief overview of the scheme and its implementation. For high-level Magma code that illustrates the entire SIDH protocol, see `SIDH.mag` in [16].

Public parameters. Together with the curve $E_0 : y^2 = x^3 + x$ and the prime $p = 2^{372}3^{239} - 1$, the public parameters are $P_A = [3^{239}](11, \sqrt{11^3 + 11})$, $Q_A = \tau(P_A)$, $P_B = [2^{372}](6, \sqrt{6^3 + 6})$, and

$Q_B = \tau(P_B)$. Given that all these square roots are in $\mathbb{F}_p$ (we choose the "odd" ones), and that Q_A and Q_B require no storage, this means that only 4 $\mathbb{F}_p$-elements (or 3004 bits) are required to fully specify the public generators. If we were to instead randomly choose extension field torsion generators without use of the distortion map, as is suggested in [19], then 16 $\mathbb{F}_p$ elements (or 12016 bits) would be required to specify the public generators.

Key generation. On input of the public parameters above, and the secret key m_A chosen as in Section 4, Alice proceeds as in Algorithm 3 (see Algorithm 2 for the simple, but slower *multiplication-based* main loop). She calls the `secret_pt` function, which computes $P_A + [m_A]Q_A$ by calling `LADDER` to compute $x([m_A]Q_A))$, before recovering the corresponding y-coordinate using the Okeya-Sakurai strategy [42]; this allows the addition of P_A and $[m_A]Q_A$. All of these operations are performed over the ground field and we proceed by taking only $x(P_A + [m_A]Q_A)$ through the main loop.

We note that our implementation requires that Alice's secret isogeny is evaluated at both of the public parameters x_{P_B} and x_{Q_B}, as well as at the x-coordinate of the difference, $x_{Q_B - P_B}$; this allows Bob to kickstart the `three_pt_ladder` function (from [19, Algorithm 1]) during his shared secret phase. Conversely, Bob must also evaluate his secret isogeny at $x_{Q_A - P_A}$. In both cases, rather than setting x_{Q-P} as a public parameter, it can be computed on-the-fly from x_P, since in this special instance, $x_{Q-P} = x_{\tau(P)-P} = i \cdot (x_P^2 + 1)/(2x_P)$. This is fed directly into our projective isogeny evaluation function, so we do not need $x_{Q-P} \in \mathbb{A}$, but can instead compute $x(Q-P) = (i(x_P^2+1) \colon 2x_P) \in \mathbb{P}^1$, which costs just one squaring and two additions in $\mathbb{F}_p$; this operation is performed with the `distort_and_diff` function.

At the conclusion of Algorithm 3, Alice outputs her public key

$$\mathrm{PK}_{\mathrm{Alice}} = [a_A, x_{\phi_A(P_B)}, x_{\phi_A(Q_B)}, x_{\phi_A(Q_B - P_B)}] \in \mathbb{F}_{p^2}^4.$$

Bob proceeds similarly, as shown in Algorithm 5 (again, see Algorithm 4 for a simpler, but slower multiplication-based approach), and outputs his public key

$$\mathrm{PK}_{\mathrm{Bob}} = [a_B, x_{\phi_B(P_A)}, x_{\phi_B(Q_A)}, x_{\phi_B(Q_A - P_A)}] \in \mathbb{F}_{p^2}^4.$$

Alice's fast key generation via Algorithm 3, using the strategies for computing the isogeny trees as given in Section 4, requires 638 multiplications-by-4 and the evaluation of 1330 4-isogenies; calling the simpler Algorithm 2 requires 17020 multiplications-by-4 an 744 4-isogeny evaluations. On Bob's side, the optimal strategy (i.e., fast key generation) requires 811 multiplications-by-3 and the evaluation of 1841 3-isogenies; the simpler version requires 28441 multiplications-by-3 and 956 3-isogeny evaluations. See Section 7 for the benchmarks and further discussion.

Shared secret. On input of $\mathrm{PK}_{\mathrm{Bob}} = [a, x_{\phi_B(P_A)}, x_{\phi_B(Q_A)}, x_{\phi_B(Q_A - P_A)}]$ and her secret key m_A, Alice calls Algorithm 7 (again, see Algorithm 6 for a more compact, but significantly slower main loop) to generate her shared secret. This starts by calling the `three_pt_ladder` function (from [19, Algorithm 1]) to compute $x(\phi_B(P_A) + [m_A]\phi_B(Q_A))$, which is used to generate the kernel of the isogeny that is computed in the main loop. Finally, Alice uses the `j_inv` function to compute her shared secret. For Bob's analogous shared key generation, see Algorithms 9 and 8.

Alice's fast key generation via Algorithm 7, again using the strategies in Section 4, requires 638 multiplications-by-4 and the evaluation of 772 4-isogenies; calling the simpler Algorithm 6 requires 17020 multiplications-by-4 and 186 4-isogeny evaluations. On Bob's side, the optimal strategy (i.e., fast key generation) requires 811 multiplications-by-3 and the evaluation of 1124 3-isogenies; the simpler version requires 28441 multiplications-by-3 and 239 3-isogeny evaluations. See Section 7 for the benchmarks and further discussion.

Remark 4. In light of the simple relationship $Q_B = \tau(P_B)$, at first glance it might seem wasteful for Alice to have to evaluate her isogeny ϕ_A at all three of x_{P_B}, x_{Q_B}, and $x_{Q_B - P_B}$. For example, the relationship $x_{Q_B} = -x_{P_B}$ suggests there might be a more efficient way to compute $\phi_A(x_{Q_B})$

directly from $\phi_A(x_{P_B})$, but this does not seem to be the case. If $\phi_A \colon E_0 \to E_A$ is Alice's secret isogeny and $\tau \colon E_0 \to E_0$ is the distortion map defined by $(x, y) \mapsto (-x, iy)$, finding $\tau' \colon E_A \to E_A$ such that $\tau' \circ \phi_A = \phi_A \circ \tau$ appears to be non-trivial. The algorithm in [23] constructs distortion maps on arbitrary supersingular curves, but using it would undoubtedly be slower than accumulating the evaluation of isogenies at x_{Q_B} through the main loop, and moreover, there is no guarantee that the distortion map that results from such a construction is the distortion map τ' we need. Equations (3) and (7) offer some evidence in this regard, e.g., there does not appear to be an exploitable relationship between their repeated application on $(X \colon Z)$ and their repeated application on $(-X \colon Z)$.

7 SIDH performance

To evaluate the performance of the proposed supersingular isogeny system and the different optimizations, we wrote a software library supporting ephemeral SIDH key-exchange. The software was mostly written in the C language and has been designed to facilitate the addition of specialized code for different platforms and applications. The first release of the library comes with a fully portable C implementation supporting 32- and 64-bit platforms and two optional x64 implementations of the field arithmetic: one implementation based on intrinsics (which is, e.g., supported on Windows OS by Visual Studio) and one implementation written in x64 assembly (which is, e.g., supported on Linux OS using GNU GCC and clang compilers). The latter two optional modules are intended for high-performance applications. All of the software is publicly available in [16].

In Table 2, we present the performance of our software using the x64 assembly implementation in comparison with the implementation proposed by [2]. Results for the implementation in [2] were obtained by benchmarking their software[12] on the same Intel Sandy Bridge and Haswell machines, running Ubuntu 14.04 LTS. Note that the results in Table 2 differ from what was presented in Table 3 in [2]. The differences might be due to the use of overclocking (i.e., TurboBoost technology). For our comparisons, we disabled TurboBoost for a more precise and fair comparison.

Table 2. Performance results (expressed in millions of clock cycles) of the proposed SIDH implementation in comparison with the implementation by Azarderakhsh et al. [2] on x64 platforms. Benchmark tests were taken with Intel's TurboBoost disabled and the results were rounded to the nearest 10^6 clock cycles. Benchmarks were done on a 3.4GHz Intel Core i7-2600 Sandy Bridge and a 3.4GHz Intel Core i7-4770 Haswell processor running Ubuntu 14.04 LTS.

Operation	This work		Prior work [2]	
	Sandy Bridge	Haswell	Sandy Bridge	Haswell
Alice's keygen	54	51	165	149
Bob's keygen	64	59	172	152
Alice's shared key	51	47	133	118
Bob's shared key	61	57	137	122
Total	230	214	608	540

Table 2 shows that the total cost of computing one Diffie-Hellman shared key (adding Alice's and Bob's individual costs together) using our software is around 2.6 and 2.5 times faster than the software from [2], on Sandy Bridge and Haswell, respectively. These results are due to the different optimizations discussed throughout this work, the most prominent two being (i) the elimination of inversions during isogeny computations by working with projective curve coefficients, and (ii) the faster modular arithmetic triggered by the selected prime and the tailor-made Montgomery reduction for SIDH primes. It is important to note that, in particular, the advantage over [2] is

[12] See https://github.com/defeo/ss-isogeny-software.

not even larger because the numerous inversions used during the isogeny computations in [2] are not computed in constant time. Making such inversions constant-time would significantly degrade their performance (see the related paragraph in Section 5).

Remark 5. In Section 4 we discussed several specialized choices that were made for reasons unrelated to performance, e.g., in the name of simplicity and/or compactness. We stress that, should future cryptanalysis reveal that these choices introduce a security vulnerability, the performance of SIDH and the performance improvements in Sections 3 and 5 are unlikely to be affected (in any meaningful way) by reverting back to the more general case(s). In particular, if it turns out that sampling from a fraction of the possible 2- and 3-torsion subgroups gives an attacker some appreciable advantage, then modifying the code to sample from the full set of torsion subgroups is merely an exercise, and the subsequent performance difference would be unnoticeable. Similarly, if the use of the base-field and trace-zero subgroups and/or the distortion map turns out to degrade SIDH security, then the main upshot of reverting to randomized public generators would be the inflated public parameters (see Section 6); the slowdown during key generation would be minor and the shared secret computations would be unchanged.

8 BigMont: a strong ECDH+SIDH hybrid

We now return to the discussion (from Section 1) of a hybrid scheme. Put simply, and in regards to both security and suitability, at present there is not enough confidence and consensus within the PQC community to warrant the standalone deployment of one particular post-quantum key exchange primitive. Subsequently, there is interest (cf. [7]) in deploying classical primitives alongside post-quantum primitives in order to hedge one's bets until a confidence-inspiring PQC key exchange standard arrives. This is particularly interesting in the case of SIDH, whose security has (because of its relatively short lifespan) received less cryptanalytic scrutiny than its post-quantum counterparts.

In this section we discuss how traditional ECDH key exchange can be included alongside SIDH key exchange at the price of a very small overhead. The main benefit of our approach is its simplicity; while SIDH could be partnered with ECDH on any of the standardized elliptic curves, this would mean that a lot more code needs to be written and/or maintained. In particular, it is often the case that the bulk of the code in high-speed ECC implementations relates to the underlying field arithmetic. Given that none of the fields underlying the standardized curves are SIDH-friendly[13], such a partnership would require either a generic implementation that would be much less efficient, or two unrelated implementations of field arithmetic. Our proposal avoids this additional complexity by performing ECDH on an elliptic curve defined over the same ground field as the one used for SIDH.

For $p = 2^{372}3^{239} - 1$, recall that our SIDH software works with isogenous curves $E_a/\mathbb{F}_{p^2}\colon y^2 = x^3 + ax^2 + x$ whose group orders are of the form $\#E_a = 2^i \cdot 3^j$, meaning that elliptic curve discrete logarithms are easy on all such curves by the Pohlig-Hellman algorithm [45]. However, there are also (exponentially many) ordinary curves of the form $E_a/\mathbb{F}_{p^2}$ that *are* cryptographically secure. In particular, over the base field $\mathbb{F}_p$, we can hope to find $a \in \mathbb{F}_p$ such that $E_a/\mathbb{F}_p$ and its quadratic twist $E'_a/\mathbb{F}_p$ are cryptographically strong, i.e., such that $E_a/\mathbb{F}_p$ is *twist-secure* [6].

Since $p \equiv 3 \bmod 4$, we searched for such a curve in exactly the same way as, e.g., Hamburg's Goldilocks curve [26] was found. Namely, since the value $(a+2)/4$ is the constant that appears in Montgomery's ladder computation [39], we searched for the value of a that gave rise to the smallest absolute value of $(a+2)/4$ (when represented as an integer in $[0, p)$), and such that $\#E_a$ and $\#E'_a$ are both 4 times a large prime. For p as above, the first such value is $a = 624450$; to make a clear distinction between curves in the supersingular isogeny class and the strong curve used to perform ECDH, we (re)label this curve as

$$M_a/\mathbb{F}_p\colon y^2 = x^3 + ax^2 + x \qquad \text{with} \qquad a = 624450.$$

[13] Nor are any of the fields large enough to support highly quantum-secure SIDH.

The trace t_{M_a} of the Frobenius endomorphism on M_a is

$$\begin{aligned} t_{M_a} =& \texttt{0x743FC8888E1D8916BAB6DD6500AD5265DFE2E04882877C} \\ & \texttt{26BA8CD28BE24D10D3E729B0BD07BC79699230B6BC69FEAC}, \end{aligned}$$

which gives $\#M_a = p + 1 - t_{M_a} = 4r_a$ and $\#M'_a = p + 1 + t_{M_a} = 4r'_a$, where r_a and r'_a are both 749-bit primes.

Following [6], every element in $\mathbb{F}_p$ corresponds to the x-coordinate of a point on either M_a or on M'_a. Together with the fact that Montgomery's `LADDER` function correctly computes underlying scalar multiplications independently of the quadratic twist, M_a being twist-secure allows us to treat all $\mathbb{F}_p$ elements as valid public keys and to perform secure ECDH without the need for any point validation.

The ECDH secret keys are integers in $[0, r_a)$. To ensure an easy constant-time `LADDER` function, we search for the smallest $\alpha \in \mathbb{N}$ such that αr_a and $(\alpha + 1)r_a - 1$ are the same bit length, which is $\alpha = 3$; accordingly, secret keys are parsed into $[3r_a, 4r_a)$ prior to the execution of scalar multiplications via `LADDER`. Subsequently, for $m \in [0, r_a)$ and $x(P) \in \mathbb{P}^1(\mathbb{F}_p)$, computing $x([m]P) =$ `LADDER`$(x(P), m, a)$ requires 1 call to `xDBL` and 750 calls to `xDBLADD` (see Table 1 for the operation counts of these functions, but note that here we can take advantage of the fixed, small constant a). As all of these computations take place over the ground field, the total time taken to compute ECDH public keys and shared secrets is only a small fraction of the total time taken to compute the analogous SIDH keys – see Table 3.

From an implementation perspective, partnering SIDH with ECDH as above is highly advantageous because all of the functions required to compute $x([m]P) =$ `LADDER`$(x(P), m, a)$ are already available from our Montgomery SIDH framework. In particular, the key generation (see Section 6) already has a tailored Montgomery `LADDER` function that works entirely over the base field, i.e., on the starting curve E_0, so computing ECDH keys is as simple as calling pre-existing functions on input of a different constant.

Though the speed overhead incurred by adding ECDH to SIDH in this way is small (see Table 3), choosing to use such a large elliptic curve group makes concatenated keys larger than they would be if a smaller elliptic curve was used for ECDH. For example, suppose we were to instead use the curve currently recommended in Suite B [41], Curve P-384, and (noting that uncompressed Curve P-384 points are larger than our proposed ECDH public keys) were to compress ECDH public keys as an x-coordinate and a sign bit. The total public key size with SIDH-compressed (resp. uncompressed) keys would then be 612 (resp. 800) bytes, instead of the 658 (resp. 845) bytes reported in Table 3. Though this difference is noticeable, it must be weighed up against the cost of the extensive additional code required to support Curve P-384, which would almost certainly share nothing in common with the existing SIDH code. Moreover, the simplicity of adding ECDH

Table 3. Comparison of standalone SIDH versus hybrid SIDH+ECDH. Timing benchmarks were taken (assuming no SIDH compression) on a 3.4GHz Intel Core i7-4770 Haswell processor running Ubuntu 14.04 LTS with TurboBoost disabled and results rounded to the nearest 10^6 clock cycles. The compression referred to is the simple compression of x_{Q-P} discussed in Section 10. For simplicity, the bit-security of the primitives was taken to be the target security level and is not intended to be precise.

comparison		standalone SIDH	hybrid SIDH+ECDH
≈ bit-security (hard problem)	classical	192 (SSDDH)	384 (ECDHP)
	PQ	128 (SSDDH)	128 (SSDDH)
public key size (bytes)	uncompressed	751	845
	compressed	564	658
speed (cc $\times 10^6$)	Alice's keygen	51	58
	Bob's keygen	59	66
	Alice's shared key	47	54
	Bob's shared key	57	64

to SIDH as we propose is not the only reason to justify slightly larger public keys; the colossal 384-bit security achieved by M_{624450} also puts it in a position to tolerate the possibility of significant future advancements in ECDLP attacks. Due to the complexity of the ECDLP on M_{624450} in comparison with all of the elliptic curves in the standards, we dub this curve "BigMont".

In Table 3 we compare hybrid SIDH+ECDH versus standalone SIDH. The take-away message is that for a less than 1.17x increase in public key sizes and less than 1.13x increase in the overall computing cost, we can increase the classical security of the key exchange from 192 bits (based on the relatively new SSDDH problem) to 384 bits (based on the long-standing ECDLP).

9 Validating public keys

Recall from Section 2 that De Feo, Jao and Plût [19] prove that SIDH is *session-key secure* (under SSDDH) in the authenticated-links adversarial model [12]. This model assumes perfectly authenticated links which effectively forces adversaries to be passive eavesdroppers; in particular, it assumes that public keys are correctly generated by honest users. While this model can be suitable for key exchange protocols that are instantiated in a truly ephemeral way, in real-world scenarios it is often the case that (static) private keys are reused. This can incentivize malicious users to create faulty public keys that allow them to learn information about the other user's static private key, and in such scenarios validating public keys becomes a mandatory practical requirement.

In traditional elliptic curve Diffie-Hellman (ECDH), validating public keys essentially amounts to checking that points are on the correct and cryptographically secure curve [9]. Such *point validation* is considered trivial in ECDH since checking that a point satisfies a curve equation requires only a handful of field multiplications and additions, and this is negligible compared to the overall cost (e.g., of a scalar multiplication).

In contexts where SIDH private keys are reused, public key validation is equally as important but is no longer as trivial. In April 2015, a group from the NSA [30] pointed out that "direct public key validation is not always possible for [...] isogeny based schemes" before describing more complicated options that validate public keys *indirectly.* In this section we describe ways to directly validate public keys that, in particular, work entirely in our compact framework, i.e., without the need of y-coordinates or of the b coefficient that fixes the Montgomery quadratic twist.

Recall from Section 6 that an honest user will generate public keys of the form

$$\mathtt{PK} = \left[a \, , \, x_P \, , \, x_Q \, , \, x_{Q-P}\right] \ \in \mathbb{F}_{p^2}^4 ,$$

where $P = (x_P, x_Q)$ and $Q = (x_Q, y_Q)$ are independent and of the same order ℓ^e on the Montgomery curve E_a that is $\mathbb{F}_{p^2}$-isogenous to E_0; the algorithms we describe below will only deem a purported public key as valid if this is indeed the case.

We first point out that, whether the public parameters are chosen from special torsion subgroups or not, the three x-coordinates in a public key will be (with overwhelming probability) defined over $\mathbb{F}_{p^2}$. The first check that must be made on public keys is that the (underlying) points P and Q are of the full order ℓ^e. If not, then an SIDH-like analogue of the Lim-Lee [34] small subgroup attack becomes a threat; e.g., an attacker could send x_Q where Q has small order q and guess the shared secret (i.e., the kernel $\langle P + [m]Q \rangle$) to learn $m \bmod q$. In addition, to our knowledge, the nature of the public key suggests that the only practical way of asserting that P and Q are independent is that their Weil or Tate pairing $e_{\ell^e}(P, Q)$ is non-trivial. If this assertion is not made, then a malicious user can simply send a public key where $Q = [\lambda]P$, which ultimately forces the shared secret to be independent of the honest party's private key[14]. Such capabilities could be catastrophic if the authentication mechanism does not detect this.

The validation of the three x-coordinates in a public key is different for Alice and Bob, so we now describe these cases separately. We then discuss how both parties validate that the first

[14] If $Q = [\lambda]P$, then the "secret" kernel generated by $R = P + [m]Q$ is the same as the kernel generated by Q and/or P, which are public.

component of the public key (i.e., a) corresponds to a supersingular curve in the correct isogeny class, and conclude the section with performance benchmarks for full validation of public keys.

Alice's validation of Bob's public key. Alice must determine whether Bob's transmission $[a, x_P, x_Q, x_R] \in \mathbb{F}_{p^2}^4$ is a valid public key. Let $n = 2^{e_A}$ and let $f_{i,P}$ be the function with divisor $\text{div}(f_{i,P}) = i(P) - ([i]P) - ([i-1]\mathcal{O})$ on E_a. We now show that through a computation related to the Weil pairing $e_n(P,Q)$, Alice can simultaneously check that x_P and x_Q correspond to the points P and Q of order n on the Montgomery curve E_a, and that P and Q are independent. At a high level, we follow the ideas of Galbraith and Lin [22] who showed that, under certain circumstances, the (traces of the) Weil and Tate pairings can be computed using only x-coordinates. One obstruction preventing the direct application of [22] in our case is that the coordinates of P and Q do not have the special (i.e., subfield) form that was needed in [22]. On the other hand, here we do not need to compute the value of the pairing, we merely need to know whether it is 1 or not. In this case we can use recursions similar to those in [22] to write the Weil pairing $e_n(P,Q) = f_{n,P}(Q)/f_{n,Q}(P)$ as an indeterminate function in y_P and y_Q. For $i = 1, \ldots, n$, we compute $f_{i,P}(Q) = \alpha_{i,P} \cdot y_P + \beta_{i,P} \cdot y_Q$ and $f_{i,Q}(P) = \alpha_{i,Q} \cdot y_P + \beta_{i,Q} \cdot y_Q$, where the subscripted α's and β's are recursively computed as functions of x_P and x_Q only, until we have

$$e_n(P,Q) = \frac{\alpha_{n,P} \cdot y_P + \beta_{n,P} \cdot y_Q}{\alpha_{n,Q} \cdot y_P + \beta_{n,Q} \cdot y_Q}.$$

It follows that $e_n(P,Q) = 1$ if and only if

$$\left(\frac{\alpha_{n,Q} - \alpha_{n,P}}{\beta_{n,P} - \beta_{n,Q}}\right)^2 = \frac{x_Q^3 + a x_Q^2 + x_Q}{x_P^3 + a x_P^2 + x_P},$$

which can be computed and checked using only x_P and x_Q. Note that no inversions are required in this validation: all point operations and line computations in Miller's algorithm are performed projectively, and the numerator and denominator of $e_n(P,Q)$ are carried separately throughout the entire computation until we cross-multiply to check the above equation.

Checking the order of the points P and Q in E_a essentially comes for free during the above computation. Since $\ell_A = 2$, the indeterminate functions $f_{n,P}$ and $f_{n,Q}$ are computed via a sequence of e_A iterations of Miller's algorithm [37], which computes both $x_{[\ell_A^{e_A}]P}$ and $x_{[\ell_A^{e_A}]Q}$ via e_A repeated calls to the `xDBL` function. For example, during the second last call that computes $(X : Z) = x([\ell_A^{e_A - 1}]P)$, we check that $Z \neq 0$, and during the last call that computes $(X' : Z') = x([\ell_A^{e_A}]P)$, we check that $Z' = 0$. If either of these checks are not satisfied (for both x_P and x_Q), then $[a, x_P, x_Q, x_R] \in \mathbb{F}_{p^2}^4$ is not a valid public key. If they are both satisfied, then we have (so far) established that x_P and x_Q correspond to independent points P and Q on the Montgomery curve $E_a/\mathbb{F}_{p^2}$, and that they have exact order $\ell_A^{e_A}$.

The fourth component of the public key can be validated via Hamburg's observation [25, §A.2], which says that $x_R = x_{Q \pm P}$ if and only if

$$4(a + x_P + x_Q + x_R) = \frac{(1 - x_P x_Q - x_P x_R - x_Q x_R)^2}{x_P x_Q x_R}.$$

Note that an adversary gains nothing from manipulating $x_R = x_{Q-P}$ into $x_R = x_{Q+P}$; both are valid possibilities for honestly generated public keys that also contain x_P and x_Q. The transmission of the *sign bit* (that chooses between these two options) during the simple compression in Section 10 is to ensure that the two parties arrive at the same shared secret.

Almost all of the computational cost in Alice's validation is the main loop of the above Weil pairing. This takes $e_A - 1 = 371$ iterations, each of which incurs $61\mathbf{M} + 12\mathbf{S} + 37\mathbf{a}$. See the function `Validate_PKB` in `Validate.mag` [16].

It remains to discuss how to check that the Montgomery curve E_a is in the correct isogeny class. Since this is done in the same way for Alice and Bob, we postpone it until after Bob's validation.

Bob's validation of Alice's public key. Bob must determine whether Alice's transmission $[a, x_P, x_Q, x_R] \in \mathbb{F}_{p^2}^4$ is a valid public key. Let $n = 3^{e_B}$, and let $f_{i,P}$ be as above. Bob could use a binary double-and-add version of Miller's algorithm to compute $f_{n,P}$, but this does not take advantage of the nature of n and would incur a significant performance penalty. We now show that it is possible to do much better, by sketching that an x-only, tripling-only version of Miller's algorithm to compute the Weil pairing can use analogous recursions as the Galbraith-Lin recursions used by Alice above.

In the case of a tripling-only version of Miller's algorithm, each iteration computes $f_{3i,P} = f_{i,P}^3 \cdot l_P/v_P$, where $\mathrm{div}(l_P/v_P) = 3([i]P) - ([3i]P) - 2(\mathcal{O})$. Thus, we take $l_P = y - (l_{2,P}x^2 + l_{1,P}x - l_{0,P})$ as the quadratic function that has respective zeros of order 3 and 1 at $[i]P$ and $(-[3i]P)$, and a pole of order 4 at $\mathcal{O}$; and, as usual, we take $v = x - x_{[3i]P}$ as the vertical function with zeros of order 1 at $[3i]P$ and $-[3i]P$ and a pole of order 2 at $\mathcal{O}$. Matching derivatives yields the $l_{i,P}$ as

$$\begin{aligned}
l_{2,P} &= y\big(6x^2 - 1 + 4ax^3 + 3x^4\big)\big/\big(8x^2(x^2 + ax + 1)^2\big),\\
l_{1,P} &= y\big(2x^2 + 4a^2x^2 + 3 + 6ax + 6ax^3 + 3x^4\big)\big/\big(4x(x^2 + ax + 1)^2\big),\\
l_{0,P} &= y\big(6x^2 + 3 + 4ax - x^4\big)\big/\big(8(x^2 + ax + 1)^2\big).
\end{aligned}$$

The key observation is that the y term appears in front of each of the l_i, which allows us to write

$$\begin{aligned}
y([3i]P) &= -\big(l_{2,P}(x([3i]P))^2 + l_{1,P}(x[3i]) + l_{0,P}\big)\\
&= y \cdot g(x([3i]P)),\\
&= y \cdot h(x([i]P)),
\end{aligned}$$

for some functions $g(x)$ and $h(x)$. This allows us to write a Montgomery tripling analogue for the doubling recurrence in [22, Lemma 2], setting $y_{[i]P} = u_i \cdot y_P$ with u_i only dependent on x-coordinates. We then carry y_P and y_Q indeterminately through Miller's algorithm, eventually checking that the pairing is non-trivial and that the points P and Q have full order $\ell_B^{e_B}$, exactly as was done for Alice[15]. Here we optimized a tailored routine for the merging of the x-only tripling function `xTPL` together with the evaluations of the (projective versions of the) line functions above.

We point out that one non-trivial difference between a "tripling-only" Weil pairing computation and the pairing computations often documented in the literature is the evaluation of functions at infinity. In particular, the final exponentiation in the Tate pairing allows evaluations of the form $f_{i,P}((Q) - (\mathcal{O}))$ to be computed as $f_{i,P}(Q)$ [4, Theorem 1]. Here the absence of a final exponentiation means that the Weil pairing, which is defined as $e_n(P,Q) = -f_{n,P}(D_Q)/f_{n,Q}(D_P)$ for divisors $D_P \sim (P) - (\mathcal{O})$ and $D_Q \sim (Q) - (\mathcal{O})$, means that we must include the evaluations of $f_{n,P}$ and $f_{n,Q}$ at $\mathcal{O}$. In our case, the computational upshot is that the evaluations of the Miller *lines* $\frac{l_P(D_Q)}{v_P(D_Q)}\big/\frac{l_Q(D_P)}{v_Q(D_P)}$ can be computed as $\frac{l_P(Q)}{v_P(Q)}\big/\frac{l_Q(P)}{v_Q(P)} \cdot \frac{l_{2,Q}}{l_{2,P}}$.

Again, almost all of the computational cost in Bob's validation is the main loop of the Weil pairing. This takes $e_B - 1 = 238$ iterations, each of which incurs $82\mathbf{M} + 16\mathbf{S} + 83\mathbf{a}$. See the function `Validate_PKA` in `Validate.mag` [16].

Validating the curve. We now show how to validate that a (i.e., the curve coefficient in Alice or Bob's public key) corresponds to a Montgomery curve E_a that is a member of the correct supersingular isogeny class. The validation has two steps: we firstly assert that $j(E_a) \notin \mathbb{F}_p$ so that E_a is not a subfield curve, then we assert that E_a is in the correct supersingular isogeny class.

For the first step, we write $a = a_0 + a_1 \cdot i$ with $a_0, a_1 \in \mathbb{F}_p$. We next write $256(a^2-3)^3 = c_0 + c_1 \cdot i$ and $a^2 - 4 = d_0 + d_1 \cdot i$ for $c_0, c_1, d_0, d_1 \in \mathbb{F}_p$. Note that a curve in Montgomery form must satisfy $a^2 - 4 \neq 0$, so either d_0 or d_1 must be non-zero. Using the p-power Frobenius automorphism, it follows from (1) that $j(E_a) \in \mathbb{F}_p$ if and only if $c_0d_1 = c_1d_0$. In this case, if $d_i \neq 0$, we have

[15] A minor difference (in our code) is that we actually check that $P' = [3^{e_B-1}]P$ and $Q' = [3^{e_B-1}]Q$ have order 3 by asserting that $x(P') = x(\texttt{xDBL}(P'))$ and $x(Q') = x(\texttt{xDBL}(Q'))$, and that neither $x(P')$ or $x(Q')$ are equivalent to $(0 : 1)$. This is because the final iteration already computes these `xDBL` operations.

$j(E_a) = c_i/d_i$ for $i \in \{0,1\}$. Thus, to assert that $j(E_a) \notin \mathbb{F}_p$, we simply compute c_0, c_1, d_0, d_1 from a and assert that $c_0 d_1 \neq c_1 d_0$ in $\mathbb{F}_p$, which totals only a handful of multiplications in $\mathbb{F}_p$. This check also catches if $a \in \mathbb{F}_p$ is submitted, i.e. $a_1 = 0$, since then $c_1 = 0 = d_1$.

The less trivial step is to validate that E_a is supersingular. To do this, we make use of Sutherland's probabilistic algorithm [53, Algorithm 1], which (for our purposes) says to pick a random point $P \in E_a(\mathbb{F}_{p^2})$, and to check whether $[p-1]P = \mathcal{O}$ or $[p+1]P = \mathcal{O}$. If this is the case, then E_a is supersingular with overwhelming probability: the probability that this test would pass if E_a was actually an ordinary curve is at most $8p/(p-1)^2 < 1/2^{747}$ [53, Proposition 1].

We now point out that E_a being supersingular is equivalent to either E_a or its quadratic twist, E'_a, belonging to the correct isogeny class. Namely, by [51, V.5.10(a)], E_a is supersingular if and only if its trace, t_{E_a}, satisfies $t_{E_a} \equiv 0 \bmod p$. Together with [55, Theorem 1], and recalling that $-2p \leq t_{E_a} \leq 2p$ [51, V.1.1], this means that there are (at most) 5 possible isogeny classes of supersingular elliptic curves, those which are described by $t_{E_a} \in \{-2p, -p, 0, p, 2p\}$. Since $p \equiv 3 \bmod 4$, there are only two possibilities for t_{E_a} that correspond to a Montgomery curve, i.e., two possible t_{E_a} such that $4 \mid \#E_a$ [39], namely $t_{E_a} = -2p$ and $t_{E_a} = 2p$. These traces respectively correspond to curves with $\#E_a = (p+1)^2$ that are in the correct isogeny class, and to curves with $\#E'_a = (p-1)^2$ that are in the isogeny class containing all of their non-trivial quadratic twists.

In our case we are trying to validate that a corresponds to a curve with $\#E_a = (p+1)^2$, so at first glance it would seem that the best route is to pick a random point $P \in E_a(\mathbb{F}_{p^2})$ and assert that $[p+1]P = \mathcal{O}$. However, generating such a random point requires a square-root computation, and it turns out that we can (again) avoid the need for a square root altogether. For a given a, recall from Section 8 (or, in turn, from [6]) that every element in $\mathbb{F}_{p^2}$ is either[16] the x-coordinate of a point on $E_a/\mathbb{F}_{p^2}$ or the x-coordinate of a point on $E'_a/\mathbb{F}_{p^2}$. This means that if E_a is supersingular, every element in $\mathbb{F}_{p^2}$ is the x-coordinate of a point whose order divides either $p-1$ or $p+1$. This gives us a way to quickly assert (with overwhelming probability) that a corresponds to a supersingular Montgomery curve in the correct isogeny class. With the Montgomery `LADDER` function as described in Section 3, we simply take a random element r in $\mathbb{F}_{p^2}$, compute $(X\colon Z) = \texttt{LADDER}((r\colon 1), a, p+1)$ and $(X'\colon Z') = \texttt{LADDER}((r\colon 1), a, p-1)$, and ensure that $Z \cdot Z' = 0$; otherwise, we reject the public key as invalid.

We can compute a condition equivalent to $Z \cdot Z' = 0$ using only one call to the `LADDER` function as follows. The condition $\mathcal{O} \in \{[p-1]P, [p+1]P\}$ is equivalent to the condition $x(P) = x([p]P)$, which can be checked by computing $(X\colon Z) = \texttt{LADDER}(x(P), a, p)$ with $x(P) = (x_P\colon 1)$ and checking that $Z \cdot x_P = X$. However, calling `LADDER` to compute $x([p]P)$ directly is undesirable; given that $p + 1 = 2^{\ell_A}3^{\ell_B}$, it is instead preferable to write a tailored ladder (consisting only of `xDBL` and `xTPL` operations) that computes a scalar multiplication by $p+1$. We do this by noting that the condition $x(P) = x([p]P)$ is equivalent to the condition that either $x([p+1]P) = x([2]P)$ or $[p+1]P = \mathcal{O}$ is satisfied.

To summarize, the validation that a corresponds to a supersingular Montgomery curve in the correct isogeny class proceeds as follows. On input of a, we choose a random element $r \in \mathbb{F}_{p^2}$ and compute $(X\colon Z) = \texttt{xDBL}((r\colon 1), a)$, we then compute $e_A - 1 = 371$ repeated doublings as $(X'\colon Z') = \texttt{xDBLe}((X\colon Z), a, 1, e_A - 1)$ before computing $e_B = 239$ repeated triplings as $(X''\colon Z'') = \texttt{xTPLe}((X'\colon Z'), a, 1, e_B)$. We then accept the curve as valid if and only if $Z''(XZ'' - X''Z) = 0$. Thus, validation of the curve amounts to 372 `xDBL` operations, 239 `xTPL` operations, and 3 subsequent $\mathbb{F}_{p^2}$ multiplications. This constitutes a small fraction of the Weil pairing computation and thus a small fraction of the full validation.

We reiterate that accepting a as valid means that one of the two quadratic twists $E_a/\mathbb{F}_{p^2}\colon y^2 = x^3 + ax^2 + x$ and $E'_a/\mathbb{F}_{p^2}\colon by^2 = x^3 + ax^2 + x$ (for b non-square in $\mathbb{F}_{p^2}$) is in the correct supersingular isogeny class. Within our framework (which ignores the choice of quadratic twist by working on the associated Kummer variety), a successful validation can be viewed as accepting such an a and then choosing to work on the quadratic twist corresponding to the correct isogeny class. For the function that validates the curve, see `test_curve` in `Validate.mag` [16].

[16] These two quadratic twists *share* points of order 2, i.e., there are (at most 3) points of the form $(x_0, 0)$ that lie on both curves; otherwise, an $\mathbb{F}_{p^2}$ element will correspond to precisely one or the other.

Remark 6. Referring back to Remark 2, we now discuss why Alice rejects Bob's public key $\texttt{PK} = [a, \dots]$ if $j(E_a) \in \mathbb{F}_p$. This is not because the subsequent shared secret would, to our knowledge, give Bob any more information about Alice's long-term private key than if $j(E_a) \in \mathbb{F}_{p^2} \backslash \mathbb{F}_p$; indeed, honestly generated public keys are assumed to be safe based on an isogeny walk that started on a subfield curve. Put another way, there is no current reason to believe that, if Bob was malicious, he would gain anything from sending a such that $j(E_a) \in \mathbb{F}_p$. Alice is rejecting Bob's public key because the (in)security of his private key is based on the difficulty of finding the isogeny between two curves defined over the ground field $\mathbb{F}_p$, for which there is a known quantum algorithm with subexponential complexity [8]. If Alice were to accept this public key as valid, both Bob's private key and therefore their shared secret could be compromised. Here the discussion is identical for Bob's validation of Alice's public key, i.e., this part of the validation is the same for both parties.

Remark 7. The speed and simplicity of the above Monte Carlo algorithm for validating E_a, as well as the ease of implementing it on the existing SIDH code base, makes it highly desirable for this stage of the public key validation. Moreover, the $O(1/p)$ chance of this procedure deeming an ordinary curve as supersingular makes it suitable for real-world deployment. We reiterate that this probability and suitability relies on the input element in $\mathbb{F}_{p^2}$ being unknown to an adversary, i.e., independent of the a value(s) it is used to validate. If an adversary knew the $\mathbb{F}_{p^2}$ element in advance, it is conceivable that this could be used to derive an ordinary curve E_a that would pass validation. On the other hand, so long as this input element is kept secret, it might be possible and desirable to safely relax the freshness/randomness requirements suggested by the algorithm. On a related note, we point out that Sutherland [53, §4] also gives a deterministic algorithm for asserting supersingularity, but its additional overhead makes it less suitable for our application.

The price of public key validation. The cost of validating Alice's and Bob's public keys is non-trivial. On our target platforms, i.e., a 3.4GHz Intel Core i7-2600 Sandy Bridge and a 3.4GHz Intel Core i7-4770 Haswell processor running Ubuntu 14.04 LTS, the validation of Alice's public key costs around 68 million and 63 million clock cycles, respectively. Similarly, the validation of Bob's public key costs around 75 million and 70 million clock cycles, respectively. Referring back to Table 2, this means that validation is between 1.17 and 1.34 times more expensive than a key generation or shared secret computation. In both cases, the cost is dominated by the expensive Weil pairing computation.

From a performance perspective, and until a much faster public key validation procedure is discovered, these results seem to suggest that instantiating SIDH in a truly ephemeral fashion (i.e., using private/public key pairs once and then discarding them) is the way to go. Indeed, the savings obtained by not having to generate fresh public keys in the static case are completely consumed by the more costly public key validation. On the other hand, in cases where performance is not the governing reason behind a static instantiation, the above public key validation is less complicated (from a protocol perspective) than the *indirect* validation in [30].

10 Compression of public keys

Recall that our public keys are of the form $[a, x_P, x_Q, x_{Q-P}] \in \mathbb{F}_{p^2}^4$, where $P = (x_P, y_P)$ and $Q = (x_Q, y_Q)$ lie on $E_a \colon y^2 = f(x) = x^3 + ax^2 + x$. We now show how to compress[17] the public key to $[a, x_P, x_Q, \texttt{bit}] \in \mathbb{F}_{p^2}^3 \times \{0, 1\}$. The idea is to use the equation from [42, Theorem 2] to write

$$2y_P y_Q = x_{Q-P} \cdot (x_Q - x_P)^2 - (x_Q + x_P)(x_Q x_P + 1) - 2a x_Q x_P. \tag{8}$$

To compress, we compute $2y_P y_Q$ from x_P, x_Q and x_{Q-P} as above and set `bit` as the corresponding *sign bit*, e.g., if $2y_P y_Q = u + vi$ with $u, v \in \mathbb{F}_p$, we can set `bit` as the lowest bit of u (viewing u

[17] This compression was derived by Ping Ngai Chung during the preparation of [15].

as an integer in $[0,p)$). To decompress, we compute $2y_Py_Q = \sqrt{4f(x_P)f(x_Q)}$, using `bit` to choose the correct square root. Rewriting the above equation reveals that x_{Q-P} can be recovered as

$$x_{Q-P} = \frac{(x_P + x_Q)(x_Px_Q + 1) + 2ax_Px_Q + 2y_Py_Q}{(x_Q - x_P)^2}.$$

In general, this would compress an $(8\log_2 p)$-bit public key into a $(6\log_2 p + 1)$-bit public key, and in our case our public keys would be compressed from 6008 bits (751 bytes) to 4507 bits (563 bytes). If computed as above, then compression is essentially free, while decompression comes at a cost of (roughly) one square root and one inversion in $\mathbb{F}_{p^2}$; assuming a valid public key (i.e., that $4f(x_P)f(x_Q)$ is indeed a square in $\mathbb{F}_{p^2}$), the former can be computed via two square roots over the base field [49, §3.3], while the latter can be computed via one inversion in the base field. This would add a small but noticeable overhead to the "shared key" computations benchmarked in Table 2.

Azarderakhsh et al. [3] recently described two techniques for key compression in isogeny-based cryptosystems. We now discuss the application of their techniques to our choice of public keys. Firstly, in [3, §3] it is remarked that the x-coordinates of points can be sent over the wire and that the sign ambiguity (when recovering the corresponding y coordinate) is immaterial, i.e., that (x, y) and $(x, -y)$ will generate the same kernel subgroup. In our case, this would suggest that we could avoid the transmission of x_{Q-P} (or the corresponding sign bit if compression is used) altogether. However, it is actually not the case that the same kernel will be generated for all four combinations of $\pm P$ and $\pm Q$. In particular, the subsequent computation $(U, V) \mapsto U + [m]V$ will yield the same subgroup if $(U, V) = (P, Q)$ or if $(U, V) = (-P, -Q)$, but will yield a different subgroup if $(U, V) = (-P, Q)$ or $(U, V) = (P, -Q)$. This means that our inclusion of x_{Q-P} (or `bit`) in the public key is necessary to guarantee that Alice and Bob generate matching shared secrets.

There are two compression techniques suggested in [3]. The first, described in [3, §3.1], shows how the curve generated under the secret isogeny can be sent over the wire using only one element in $\mathbb{F}_{p^2}$, rather than, e.g., the two coefficients used to specify a short Weierstrass curve. We note that this technique does not help in our case, since our framework (which unifies a Montgomery curve with its twist) already allows for the "curve part" of the public keys to be specified using one element $a \in \mathbb{F}_{p^2}$. We note that our approach has the advantage of avoiding conversions between curve models as well as the other overheads discussed in [3, §3.1].

On the other hand, the second technique in [3, §3.2] could be used to further compress our public keys. Rather than sending x_P and x_Q over the wire, Azarderakhsh et al. suggest that each of these *points* could instead be transferred as their scalar representations with respect to a fixed torsion basis $\{R, S\}$. In particular, if $\{R, S\}$ is a basis for $E[\ell^e]$, rather than sending x_P (which corresponds to $\pm P \in E[\ell^e]$), we could compute α and β in $\mathbb{Z}/\ell^e\mathbb{Z}$ such that $[\alpha]R + [\beta]S = P$ and transmit (α, β) instead. In general this technique allows x_P and x_Q to be transferred using roughly half as many bits (the exact reduction depends on the specific curve and its torsion sizes – see [3]), but there is a significant overhead in doing so. Specifically, while decompression essentially amounts to recovering P via the computation of $[\alpha]R + [\beta]S$ (and this small overhead would probably be worthwhile), the compression phase involves solving a smooth order discrete logarithm problem via the Pohlig-Hellman algorithm [45]. The benchmarks in [3, §5.4] show that this part of the compression alone costs over 10 times the total cost of the entire key exchange protocol. In our case, and presuming that x_{Q-P} is compressed as above, this further compression would reduce our 4507-bit public keys to 3005-bit public keys. Given the drastic loss in performance, our software does not incorporate this Pohlig-Hellman-style compression, but (as is discussed in [3, §6]) we note that there could be situations where this further compression makes the difference between the applicability (or not) of isogeny-based key exchange, e.g., when the target application has a strict upper bound on the size of public keys.

11 Conclusion

We presented several new algorithms that have given rise to more efficient SIDH key exchange. We built a software library around a supersingular isogeny class determined by a fixed base curve that was chosen to target 128 bits of quantum security, and showed that these techniques give rise to a factor speedup above 2.5x over the previous fastest SIDH software. To our knowledge, our SIDH key exchange software is the first such implementation to run in constant time, and offers a range of additional benefits, such as compactness. In addition, we introduced two new techniques that bridge the gap between theoretical and real-world deployment of SIDH key exchange: the ECDH+SIDH hybrid and an algorithm for full validation of public keys. The speed of our software (and the size of the public keys it generates) highlights the potential that SIDH currently offers as a candidate for post-quantum key exchange.

References

1. D. F. Aranha, K. Karabina, P. Longa, C. H. Gebotys, and J. López. Faster explicit formulas for computing pairings over ordinary curves. In *Advances in Cryptology – Eurocrypt 2011*, volume 6632 of *Lecture Notes in Computer Science*, pages 48–68. Springer, 2011.
2. R. Azarderakhsh, D. Fishbein, and D. Jao. Efficient implementations of a quantum-resistant key-exchange protocol on embedded systems. Technical report, 2014, URL: http://cacr.uwaterloo.ca/techreports/2014/cacr2014-20.pdf.
3. R. Azarderakhsh, D. Jao, K. Kalach, B. Koziel, and C. Leonardi. Key compression for isogeny-based cryptosystems. To appear in ASIA PKC 2016, preprint: http://eprint.iacr.org/2016/229, 2016.
4. P. S. L. M. Barreto, B. Lynn, and M. Scott. Efficient implementation of pairing-based cryptosystems. *J. Cryptology*, 17(4):321–334, 2004.
5. P. Barrett. Implementing the Rivest Shamir and Adleman public key encryption algorithm on a standard digital signal processor. In A. M. Odlyzko, editor, *CRYPTO '86*, volume 263 of *LNCS*, pages 311–323. Springer, 1986.
6. D. J. Bernstein. Curve25519: New Diffie-Hellman speed records. In M. Yung, Y. Dodis, A. Kiayias, and T. Malkin, editors, *Public Key Cryptography - PKC 2006, 9th International Conference on Theory and Practice of Public-Key Cryptography, New York, NY, USA, April 24-26, 2006, Proceedings*, volume 3958 of *Lecture Notes in Computer Science*, pages 207–228. Springer, 2006.
7. D. J. Bernstein. The post-quantum internet. Invited talk at PQCrypto 2016: https://cr.yp.to/talks/2016.02.24/slides-djb-20160224-a4.pdf, Feb, 2016.
8. J. Biasse, D. Jao, and A. Sankar. A quantum algorithm for computing isogenies between supersingular elliptic curves. In W. Meier and D. Mukhopadhyay, editors, *INDOCRYPT 2014*, volume 8885 of *LNCS*, pages 428–442. Springer, 2014.
9. I. Biehl, B. Meyer, and V. Müller. Differential fault attacks on elliptic curve cryptosystems. In M. Bellare, editor, *CRYPTO 2000*, volume 1880 of *LNCS*, pages 131–146. Springer, 2000.
10. W. Bosma, J. Cannon, and C. Playoust. The Magma algebra system I: The user language. *J. Symbolic Computation*, 24(3):235–265, 1997.
11. R. Bröker. Constructing supersingular elliptic curves. *J. Comb. Number Theory*, 1(3):269–273, 2009.
12. R. Canetti and H. Krawczyk. Analysis of key-exchange protocols and their use for building secure channels. In B. Pfitzmann, editor, *EUROCRYPT 2001*, volume 2045 of *LNCS*, pages 453–474. Springer, 2001.
13. L. Chen, S. Jordan, Y.-K. Liu, D. Moody, R. Peralta, R. Perlner, and D. Smith-Tone. Report on post-quantum cryptography. NISTIR 8105, DRAFT, 2016. http://csrc.nist.gov/publications/drafts/nistir-8105/nistir_8105_draft.pdf.
14. A. M. Childs, D. Jao, and V. Soukharev. Constructing elliptic curve isogenies in quantum subexponential time. *J. Mathematical Cryptology*, 8(1):1–29, 2014.
15. P. N. Chung, C. Costello, and B. Smith. Fast, uniform, and compact scalar multiplication for elliptic curves and genus 2 Jacobians with applications to signature schemes. Cryptology ePrint Archive, Report 2015/983, 2015. http://eprint.iacr.org/.
16. C. Costello, P. Longa, and M. Naehrig. SIDH Library. http://research.microsoft.com/en-us/downloads/bd5fd4cd-61b6-458a-bd94-b1f406a3f33f/, 2016.
17. C. Delfs and S. D. Galbraith. Computing isogenies between supersingular elliptic curves over $\mathbb{F}_p$. *Des. Codes Cryptography*, 78(2):425–440, 2016.

18. M. H. Devoret and R. J. Schoelkopf. Superconducting circuits for quantum information: an outlook. *Science*, 339(6124):1169–1174, 2013.
19. 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.
20. S. D. Galbraith. Constructing isogenies between elliptic curves over finite fields. *LMS Journal of Computation and Mathematics*, 2:118–138, 1999.
21. S. D. Galbraith, F. Hess, and N. P. Smart. Extending the GHS weil descent attack. In L. R. Knudsen, editor, *EUROCRYPT 2002*, volume 2332 of *LNCS*, pages 29–44. Springer, 2002.
22. S. D. Galbraith and X. Lin. Computing pairings using x -coordinates only. *Des. Codes Cryptography*, 50(3):305–324, 2009.
23. S. D. Galbraith and V. Rotger. Easy decision Diffie-Hellman groups. *LMS Journal of Computation and Mathematics*, 7:201–218, 1 2004.
24. S. D. Galbraith and A. Stolbunov. Improved algorithm for the isogeny problem for ordinary elliptic curves. *Appl. Algebra Eng. Commun. Comput.*, 24(2):107–131, 2013.
25. M. Hamburg. Fast and compact elliptic-curve cryptography. *IACR Cryptology ePrint Archive*, 2012:309, 2012.
26. M. Hamburg. Ed448-Goldilocks, a new elliptic curve. Cryptology ePrint Archive, Report 2015/625, 2015. http://eprint.iacr.org/.
27. 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.
28. D. Jao and L. De Feo. Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies. In B. Yang, editor, *PQCrypto 2011*, volume 7071 of *LNCS*, pages 19–34. Springer, 2011.
29. J. Kelly, R. Barends, A. G. Fowler, A. Megrant, E. Jeffrey, T. C. White, D. Sank, J. Y. Mutus, B. Campbell, Yu Chen, Z. Chen, B. Chiaro, A. Dunsworth, I.-C. Hoi, C. Neill, P. J. J. O/'Malley, C. Quintana, P. Roushan, A. Vainsencher, J. Wenner, A. N. Cleland, and John M. Martinis. State preservation by repetitive error detection in a superconducting quantum circuit. *Nature*, 519:66–69, 2015.
30. 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.
31. C. K. Koc, T. Acar, and B. S. Kaliski Jr. Analyzing and comparing Montgomery multiplication algorithms. *IEEE Micro*, 16(3):26–33, 1996.
32. P. C. Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In N. Koblitz, editor, *CRYPTO '96*, volume 1109 of *LNCS*, pages 104–113. Springer, 1996.
33. A. K. Lenstra. Generating RSA moduli with a predetermined portion. In Kazuo Ohta and Dingyi Pei, editors, *ASIACRYPT '98*, volume 1514 of *LNCS*, pages 1–10. Springer, 1998.
34. C. H. Lim and P. J. Lee. A key recovery attack on discrete log-based schemes using a prime order subgroupp. In B. S. Kaliski Jr., editor, *CRYPTO '97*, volume 1294 of *LNCS*, pages 249–263. Springer, 1997.
35. R. J. McEliece. A public-key cryptosystem based on algebraic coding theory. *Coding Thv*, 4244:114–116, 1978.
36. R. C. Merkle. *Secrecy, authentication, and public key systems.* PhD thesis, Stanford University, 1979.
37. V. S. Miller. The Weil pairing, and its efficient calculation. *J. Cryptology*, 17(4):235–261, 2004.
38. P. L. Montgomery. Modular multiplication without trial division. *Mathematics of computation*, 44(170):519–521, 1985.
39. P. L. Montgomery. Speeding the Pollard and elliptic curve methods of factorization. *Mathematics of computation*, 48(177):243–264, 1987.
40. M. Mosca. Cybersecurity in an era with quantum computers: will we be ready? Cryptology ePrint Archive, Report 2015/1075, 2015. http://eprint.iacr.org/.
41. National Security Agency (NSA). Cryptography today. August 2015. https://www.nsa.gov/ia/programs/suiteb_cryptography/.
42. K. Okeya and K. Sakurai. Efficient elliptic curve cryptosystems from a scalar multiplication algorithm with recovery of the y-coordinate on a Montgomery-form elliptic curve. In C. K. Koç, D. Naccache, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2001, Third International Workshop, Paris, France, May 14-16, 2001, Proceedings*, volume 2162 of *Lecture Notes in Computer Science*, pages 126–141. Springer, 2001.
43. D. Page. Theoretical use of cache memory as a cryptanalytic side-channel. Technical report CSTR-02-003, Department of Computer Science, University of Bristol, 2002. http://www.cs.bris.ac.uk/Publications/Papers/1000625.pdf.

44. J. Patarin. Hidden fields equations (HFE) and isomorphisms of polynomials (IP): two new families of asymmetric algorithms. In U. M. Maurer, editor, *EUROCRYPT '96*, volume 1070 of *LNCS*, pages 33–48. Springer, 1996.
45. S. C. Pohlig and M. E. Hellman. An improved algorithm for computing logarithms over GF(p) and its cryptographic significance. *Information Theory, IEEE Transactions on*, 24(1):106–110, 1978.
46. A. Rostovtsev and A. Stolbunov. Public-key cryptosystem based on isogenies. Cryptology ePrint Archive, Report 2006/145, 2006. http://eprint.iacr.org/.
47. M. Scott. Fast machine code for modular multiplication, 1995. Manuscript, available for download at ftp://ftp.computing.dcu.ie/pub/crypto/fast mod mult2.ps.
48. M. Scott. Computing the Tate pairing. In A. Menezes, editor, *CT-RSA 2005*, volume 3376 of *LNCS*, pages 293–304. Springer, 2005.
49. M. Scott. Implementing cryptographic pairings. In T. Takagi, T. Okamoto, E. Okamoto, and T. Okamoto, editors, *Pairing 2007*, volume 4575 of *LNCS*, pages 177–196. Springer, 2007.
50. 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.
51. J. H. Silverman. *The Arithmetic of Elliptic Curves, 2nd Edition.* Graduate Texts in Mathematics. Springer, 2009.
52. A. Stolbunov. *Cryptographic Schemes Based on Isogenies.* PhD thesis, Norwegian University of Science and Technology, 2012. http://www.item.ntnu.no/_media/people/personalpages/phd/anton/stolbunov-crypthographic_schemes_based_on_isogenies-phd_thesis_2012.pdf.
53. A. V. Sutherland. Identifying supersingular elliptic curves. *LMS Journal of Computation and Mathematics*, 15:317–325, 2012.
54. S. Tani. Claw finding algorithms using quantum walk. *Theor. Comput. Sci.*, 410(50):5285–5297, 2009.
55. J. Tate. Endomorphisms of abelian varieties over finite fields. *Inventiones mathematicae*, 2(2):134–144, 1966.
56. J. Vélu. Isogénies entre courbes elliptiques. *CR Acad. Sci. Paris Sér. AB*, 273:A238–A241, 1971.
57. E. R. Verheul. Evidence that XTR is more secure than supersingular elliptic curve cryptosystems. *J. Cryptology*, 17(4):277–296, 2004.
58. C. D. Walter. Montgomery exponentiation needs no final subtractions. *Electronics Letters*, 35(21):1831–1832, 1999.
59. S. Zhang. Promised and distributed quantum search. In L. Wang, editor, *COCOON 2005*, volume 3595 of *LNCS*, pages 430–439. Springer, 2005.

A Key exchange algorithms

Here we describe eight algorithms for SIDH key exchange, which correspond to the following scenarios:

- Algorithm 2: Alice's key generation using the simple (but slow) *multiplication-based* strategy for traversing the isogeny tree.
- Algorithm 3: Alice's key generation using an optimal strategy for traversing the isogeny tree.
- Algorithm 4: Bob's key generation using the simple (but slow) *multiplication-based* strategy for traversing the isogeny tree.
- Algorithm 5: Bob's key generation using an optimal strategy for traversing the isogeny tree.
- Algorithm 6: Alice's shared secret computation using the simple (but slow) *multiplication-based* strategy for traversing the isogeny tree.
- Algorithm 7: Alice's shared secret computation using an optimal strategy for traversing the isogeny tree.
- Algorithm 8: Bob's shared secret computation using the simple (but slow) *multiplication-based* strategy for traversing the isogeny tree.
- Algorithm 9: Bob's shared secret computation using an optimal strategy for traversing the isogeny tree.

All of these high-level descriptions can also be found in `SIDH.mag` in [16].

Algorithm 2 Simple key generation for Alice. (See `keygen_Alice_simple` in `SIDH.mag` in [16].)

Input: Public parameters $x_{P_B}, x_{P_A}, y_{P_A}$ and Alice's secret key $\mathrm{SK}_{\mathrm{Alice}} = m_A \in \{1, 2, \ldots, \ell_A^{e_A-1} - 1\} \cdot \ell_A$.
Output: Alice's public key $\mathrm{PK}_{\mathrm{Alice}} = [a_A, x_{\phi_A(P_B)}, x_{\phi_A(Q_B)}, x_{\phi_A(Q_B-P_B)}]$, where $\phi_A : E_0 \to E_{a_A}$ is the secret isogeny with kernel generated by $P_A + [m_A]Q_A$ and $E_{a_A} : y^2 = x^3 + a_A x^2 + x$.

```
$A \leftarrow 0,\ C \leftarrow 1$
$X_P \leftarrow x_{P_B}, Z_P \leftarrow 1$
$X_Q \leftarrow -x_{P_B}, Z_Q \leftarrow 1$
$X_D, Z_D \leftarrow \texttt{distort\_and\_diff}(x_{P_B})$
$X_{R,0}, X_{R,1}, Z_R \leftarrow \texttt{secret\_pt}(x_{P_A}, y_{P_A}, \mathrm{SK}_{\mathrm{Alice}})$
$X_R \leftarrow X_{R,0} + X_{R,1} i$
$X_P, Z_P \leftarrow \texttt{first\_4\_isog}(X_P, Z_P, A)$
$X_Q, Z_Q \leftarrow \texttt{first\_4\_isog}(X_Q, Z_Q, A)$
$X_D, Z_D \leftarrow \texttt{first\_4\_isog}(X_D, Z_D, A)$
$X_R, X_Z, A, C \leftarrow \texttt{first\_4\_isog}(X_R, Z_R, A)$
for $e = e_A - 4$ to $0$ by $-2$ do
    $X_S, Z_S \leftarrow \texttt{xDBLe}(X_R, Z_R, A, C, e)$
    $A, C, \mathrm{consts} \leftarrow \texttt{get\_4\_isog}(X_S, Z_S)$
    $X_R, Z_R \leftarrow \texttt{eval\_4\_isog}(\mathrm{consts}, X_R, Z_R)$
    $X_P, Z_P \leftarrow \texttt{eval\_4\_isog}(\mathrm{consts}, X_P, Z_P)$
    $X_Q, Z_Q \leftarrow \texttt{eval\_4\_isog}(\mathrm{consts}, X_Q, Z_Q)$
    $X_D, Z_D \leftarrow \texttt{eval\_4\_isog}(\mathrm{consts}, X_D, Z_D)$
end for
$C, Z_P, Z_Q, Z_D \leftarrow \texttt{inv\_4\_way}(C, Z_P, Z_Q, Z_D)$
$A \leftarrow A \cdot C$
$x_P \leftarrow X_P \cdot Z_P$
$x_Q \leftarrow X_Q \cdot Z_Q$
$x_D \leftarrow X_D \cdot Z_D$
return $\mathrm{PK}_{\mathrm{Alice}} = [A, x_P, x_Q, x_D]$
```

Algorithm 3 Fast key generation for Alice using an optimal strategy for isogeny computation and evaluation. (See `keygen_Alice_fast` in `SIDH.mag` in [16].)

Input: Public parameters $x_{P_B}, x_{P_A}, y_{P_A}$, a list of positive integers L_A of length $f_A = e_A/2 - 1$ describing the isogeny-graph strategy and Alice's secret key $\mathrm{SK}_{\mathrm{Alice}} = m_A \in \{1, 2, \ldots, \ell_A^{e_A-1} - 1\} \cdot \ell_A$.

Output: Alice's public key $\mathrm{PK}_{\mathrm{Alice}} = [a_A, x_{\phi_A(P_B)}, x_{\phi_A(Q_B)}, x_{\phi_A(Q_B - P_B)}]$, where $\phi_A : E_0 \to E_{a_A}$ is the secret isogeny with kernel generated by $P_A + [m_A]Q_A$ and $E_{a_A} : y^2 = x^3 + a_A x^2 + x$.

```
$A \leftarrow 0,\ C \leftarrow 1$
$X_P \leftarrow x_{P_B}, Z_P \leftarrow 1$
$X_Q \leftarrow -x_{P_B}, Z_Q \leftarrow 1$
$X_D, Z_D \leftarrow$ distort_and_diff$(x_{P_B})$
$X_{R,0}, X_{R,1}, Z_R \leftarrow$ secret_pt$(x_{P_A}, y_{P_A}, \mathrm{SK}_{\mathrm{Alice}})$
$X_R \leftarrow X_{R,0} + X_{R,1} i$
$X_P, Z_P \leftarrow$ first_4_isog$(X_P, Z_P, A)$
$X_Q, Z_Q \leftarrow$ first_4_isog$(X_Q, Z_Q, A)$
$X_D, Z_D \leftarrow$ first_4_isog$(X_D, Z_D, A)$
$X_R, Z_R, A, C \leftarrow$ first_4_isog$(X_R, Z_R, A)$
pts = [ ], $i = 0$
for $j = 1$ to $f_A$ do
    while $i < f_A - j$ do
        Push $(X_R, Z_R, i)$ to pts, i.e. append it to the end of pts
        $e \leftarrow L_A[f_A - i - j + 1]$
        $X_R, Z_R =$ xDBLe$(X_R, Z_R, A, C, 2e)$
        $i \leftarrow i + e$
    end while
    $A, C, \mathrm{consts} \leftarrow$ get_4_isog$(X_S, Z_S)$
    for $S = (X_S, Z_S, i_S) \in$ pts do
        $X_S, Z_S \leftarrow$ eval_4_isog$(\mathrm{consts}, X_S, Z_S)$
    end for
    $X_P, Z_P \leftarrow$ eval_4_isog$(\mathrm{consts}, X_P, Z_P)$
    $X_Q, Z_Q \leftarrow$ eval_4_isog$(\mathrm{consts}, X_Q, Z_Q)$
    $X_D, Z_D \leftarrow$ eval_4_isog$(\mathrm{consts}, X_D, Z_D)$
    Pop $(X_S, Z_S, i_S)$ from pts, i.e. remove it from the end of pts
    $X_R \leftarrow X_S,\ Z_R \leftarrow Z_S,\ i \leftarrow i_S$
end for
$A, C, \mathrm{consts} \leftarrow$ get_4_isog$(X_S, Z_S)$
$X_P, Z_P \leftarrow$ eval_4_isog$(\mathrm{consts}, X_P, Z_P)$
$X_Q, Z_Q \leftarrow$ eval_4_isog$(\mathrm{consts}, X_Q, Z_Q)$
$X_D, Z_D \leftarrow$ eval_4_isog$(\mathrm{consts}, X_D, Z_D)$
$C, Z_P, Z_Q, Z_D \leftarrow$ inv_4_way$(C, Z_P, Z_Q, Z_D)$
$A \leftarrow A \cdot C$
$x_P \leftarrow X_P \cdot Z_P$
$x_Q \leftarrow X_Q \cdot Z_Q$
$x_D \leftarrow X_D \cdot Z_D$
return $\mathrm{PK}_{\mathrm{Alice}} = [A, x_P, x_Q, x_D]$
```

Algorithm 4 Simple key generation for Bob. (See keygen_Bob_simple in SIDH.mag in [16].)

Input: Public parameters $x_{P_A}, x_{P_B}, y_{P_B}$ and Bob's secret key $\mathrm{SK}_{\mathrm{Bob}} = m_B \in \{1, 2, \ldots, \ell_B^{e_B-1} - 1\} \cdot \ell_B$.
Output: Bob's public key $\mathrm{PK}_{\mathrm{Bob}} = [a_B, x_{\phi_B(P_A)}, x_{\phi_B(Q_A)}, x_{\phi_B(Q_A-P_A)}]$, where $\phi_B : E_0 \rightarrow E_{a_B}$ is the secret isogeny with kernel generated by $P_B + [m_B]Q_B$ and $E_{a_B} : y^2 = x^3 + a_B x^2 + x$.

```
$A \leftarrow 0,\ C \leftarrow 1$
$X_P \leftarrow x(P_A), Z_P \leftarrow 1$
$X_Q \leftarrow x(Q_A), Z_Q \leftarrow 1$
$X_D, Z_D \leftarrow$ distort_and_diff($x_{P_A}$)
$X_{R,0}, X_{R,1}, Z_R \leftarrow$ secret_pt($x_{P_B}, y_{P_B}, \mathrm{SK}_{\mathrm{Bob}}$)
$X_R \leftarrow X_{R,0} + X_{R,1} i$
for $e = e_B - 1$ to $0$ by $-1$ do
    $X_S, Z_S \leftarrow$ xTPLe($X_R, Z_R, A, C, e$)
    $A, C, \mathrm{consts} \leftarrow$ get_3_isog($X_S, Z_S$)
    $X_R, Z_R \leftarrow$ eval_3_isog($\mathrm{consts}, X_R, Z_R$)
    $X_P, Z_P \leftarrow$ eval_3_isog($\mathrm{consts}, X_P, Z_P$)
    $X_Q, Z_Q \leftarrow$ eval_3_isog($\mathrm{consts}, X_Q, Z_Q$)
    $X_D, Z_D \leftarrow$ eval_3_isog($\mathrm{consts}, X_D, Z_D$)
end for
$C, Z_P, Z_Q, Z_D \leftarrow$ inv_4_way($C, Z_P, Z_Q, Z_D$)
$A \leftarrow A \cdot C$
$x_P \leftarrow X_P \cdot Z_P$
$x_Q \leftarrow X_Q \cdot Z_Q$
$x_D \leftarrow X_D \cdot Z_D$
return $\mathrm{PK}_{\mathrm{Bob}} = [A, x_P, x_Q, x_D]$
```

Algorithm 5 Fast key generation for Bob using an optimal strategy for isogeny computation and evaluation. (See `keygen_Bob_fast` in `SIDH.mag` in [16].)

Input: Public parameters $x_{P_A}, x_{P_B}, y_{P_B}$, a list of positive integers L_B of length e_B describing the isogeny-graph strategy and Bob's secret key $\mathrm{SK}_{\mathrm{Bob}} = m_B \in \{1, 2, \ldots, \ell_B^{e_B-1} - 1\} \cdot \ell_B$.
Output: Bob's public key $\mathrm{PK}_{\mathrm{Bob}} = [a_B, x_{\phi_B(P_A)}, x_{\phi_B(Q_A)}, x_{\phi_B(Q_A-P_A)}]$, where $\phi_B : E_0 \to E_{a_B}$ is the secret isogeny with kernel generated by $P_B + [m_B]Q_B$ and $E_{a_B} : y^2 = x^3 + a_B x^2 + x$.

```
$A \leftarrow 0,\ C \leftarrow 1$
$X_P \leftarrow x(P_A), Z_P \leftarrow 1$
$X_Q \leftarrow x(Q_A), Z_Q \leftarrow 1$
$X_D, Z_D \leftarrow \texttt{distort\_and\_diff}(x_{P_A})$
$X_{R,0}, X_{R,1}, Z_R \leftarrow \texttt{secret\_pt}(x_{P_B}, y_{P_B}, \mathrm{SK}_{\mathrm{Bob}})$
$X_R \leftarrow X_{R,0} + X_{R,1}i$
pts = [ ], $i = 0$
for $j = 1$ to $e_B$ do
    while $i < e_B - j$ do
        Push $(X_R, Z_R, i)$ to pts, i.e. append it to the end of pts
        $e \leftarrow L_B[e_B - i - j + 1]$
        $X_R, Z_R = \texttt{xTPLe}(X_R, Z_R, A, C, e)$
        $i \leftarrow i + e$
    end while
    $A, C, \mathrm{consts} \leftarrow \texttt{get\_3\_isog}(X_S, Z_S)$
    for $S = (X_S, Z_S, i_S) \in$ pts do
        $X_S, Z_S \leftarrow \texttt{eval\_3\_isog}(\mathrm{consts}, X_S, Z_S)$
    end for
    $X_P, Z_P \leftarrow \texttt{eval\_3\_isog}(\mathrm{consts}, X_P, Z_P)$
    $X_Q, Z_Q \leftarrow \texttt{eval\_3\_isog}(\mathrm{consts}, X_Q, Z_Q)$
    $X_D, Z_D \leftarrow \texttt{eval\_3\_isog}(\mathrm{consts}, X_D, Z_D)$
    Pop $(X_S, Z_S, i_S)$ from pts, i.e. remove it from the end of pts
    $X_R \leftarrow X_S,\ Z_R \leftarrow Z_S,\ i \leftarrow i_S$
end for
$A, C, \mathrm{consts} \leftarrow \texttt{get\_4\_isog}(X_S, Z_S)$
$X_P, Z_P \leftarrow \texttt{eval\_3\_isog}(\mathrm{consts}, X_P, Z_P)$
$X_Q, Z_Q \leftarrow \texttt{eval\_3\_isog}(\mathrm{consts}, X_Q, Z_Q)$
$X_D, Z_D \leftarrow \texttt{eval\_3\_isog}(\mathrm{consts}, X_D, Z_D)$
$C, Z_P, Z_Q, Z_D \leftarrow \texttt{inv\_4\_way}(C, Z_P, Z_Q, Z_D)$
$A \leftarrow A \cdot C$
$x_P \leftarrow X_P \cdot Z_P$
$x_Q \leftarrow X_Q \cdot Z_Q$
$x_D \leftarrow X_D \cdot Z_D$
return $\mathrm{PK}_{\mathrm{Alice}} = [A, x_P, x_Q, x_D]$
```

Algorithm 6 Simple shared secret algorithm for Alice. (See `shared_secret_Alice_simple` in `SIDH.mag` in [16].)

Input: Bob's public key $\mathrm{PK}_{\mathrm{Bob}} = [a_B, x_{\phi_B(P_A)}, x_{\phi_B(Q_A)},\ x_{\phi_B(Q_A-P_A)}]$ and Alice's secret key $\mathrm{SK}_{\mathrm{Alice}} = m_A \in \{1, 2, \ldots, \ell_A^{e_A-1} - 1\} \cdot \ell_A$.

Output: A shared secret j-invariant of an elliptic curve $E_{\mathrm{shared}} : y^2 = x^3 + (A/C)x^2 + x$.

```
$A \leftarrow \mathrm{PK}_{\mathrm{Bob}}[1],\ C \leftarrow 1$
$X_R, Z_R \leftarrow$ LADDER_3_pt$(\mathrm{SK}_{\mathrm{Alice}}, \mathrm{PK}_{\mathrm{Bob}}[2], \mathrm{PK}_{\mathrm{Bob}}[3], \mathrm{PK}_{\mathrm{Bob}}[4], A)$
$X_R, Z_R, A, C \leftarrow$ first_4_isog$(X_R, Z_R, A)$
for $e = e_A - 4$ to $2$ by $-2$ do
    $X_S, Z_S \leftarrow$ xDBLe$(X_R, Z_R, A, C, e)$
    $A, C, \mathrm{consts} \leftarrow$ get_4_isog$(X_S, Z_S)$
    $X_R, Z_R \leftarrow$ eval_4_isog$(\mathrm{consts}, X_R, Z_R)$
end for
$A, C \leftarrow$ get_4_isog$(X_R, Z_R)$
return j_inv$(A, C)$
```

Algorithm 7 Fast shared secret algorithm for Alice. (See `shared_secret_Alice_fast` in `SIDH.mag` in [16].)

Input: Bob's public key $\mathrm{PK}_{\mathrm{Bob}} = [a_B, x_{\phi_B(P_A)}, x_{\phi_B(Q_A)},\ x_{\phi_B(Q_A-P_A)}]$, a list of positive integers L_A of length $f_A = e_A/2 - 1$ describing the isogeny-graph strategy and Alice's secret key $\mathrm{SK}_{\mathrm{Alice}} = m_A \in \{1, 2, \ldots, \ell_A^{e_A-1} - 1\} \cdot \ell_A$.

Output: A shared secret j-invariant of an elliptic curve $E_{\mathrm{shared}} : y^2 = x^3 + (A/C)x^2 + x$.

```
$A \leftarrow \mathrm{PK}_{\mathrm{Bob}}[1],\ C \leftarrow 1$
$X_R, Z_R \leftarrow$ LADDER_3_pt$(\mathrm{SK}_{\mathrm{Alice}}, \mathrm{PK}_{\mathrm{Bob}}[2], \mathrm{PK}_{\mathrm{Bob}}[3], \mathrm{PK}_{\mathrm{Bob}}[4], A)$
$X_R, Z_R, A, C \leftarrow$ first_4_isog$(X_R, Z_R, A)$
pts = [ ], $i = 0$
for $j = 1$ to $f_A$ do
    while $i < f_A - j$ do
        Push $(X_R, Z_R, i)$ to pts, i.e. append it to the end of pts
        $e \leftarrow L_A[f_A - i - j + 1]$
        $X_R, Z_R =$ xDBLe$(X_R, Z_R, A, C, 2e)$
        $i \leftarrow i + e$
    end while
    $A, C, \mathrm{consts} \leftarrow$ get_4_isog$(X_S, Z_S)$
    for $S = (X_S, Z_S, i_S) \in$ pts do
        $X_S, Z_S \leftarrow$ eval_4_isog$(\mathrm{consts}, X_S, Z_S)$
    end for
    Pop $(X_S, Z_S, i_S)$ from pts, i.e. remove it from the end of pts
    $X_R \leftarrow X_S,\ Z_R \leftarrow Z_S,\ i \leftarrow i_S$
end for
$A, C \leftarrow$ get_4_isog$(X_R, Z_R)$
return j_inv$(A, C)$
```

Algorithm 8 Simple shared secret algorithm for Bob. (See `shared_secret_Bob_simple` in `SIDH.mag` in [16].)

Input: Alice's public key $\mathrm{PK}_{\mathrm{Alice}} = [a_A, x_{\phi_A(P_B)}, x_{\phi_A(Q_B)},\ x_{\phi_A(Q_B-P_B)}$ and Bob's secret key $\mathrm{SK}_{\mathrm{Bob}} = m_B \in \{1, 2, \ldots, \ell_B^{e_B-1} - 1\} \cdot \ell_B$.

Output: A shared secret j-invariant of an elliptic curve $E_{\mathrm{shared}} : y^2 = x^3 + (A/C)x^2 + x$.

```
$A \leftarrow \mathrm{PK}_{\mathrm{Alice}}[1],\ C \leftarrow 1$
$X_R, Z_R \leftarrow$ LADDER_3_pt$(\mathrm{SK}_{\mathrm{Bob}}, \mathrm{PK}_{\mathrm{Alice}}[2], \mathrm{PK}_{\mathrm{Alice}}[3], \mathrm{PK}_{\mathrm{Alice}}[4], A)$
for $e = e_B - 1$ to $1$ by $-1$ do
    $X_S, Z_S \leftarrow$ xTPLe$(X_R, Z_R, A, C, e)$
    $A, C, \mathrm{consts} \leftarrow$ get_3_isog$(X_S, Z_S)$
    $X_R, Z_R \leftarrow$ eval_3_isog$(\mathrm{consts}, X_R, Z_R)$
end for
$A, C, \mathrm{consts} \leftarrow$ get_3_isog$(X_R, Z_R)$
return j_inv$(A, C)$
```

Algorithm 9 Fast shared secret algorithm for Bob. (See `shared_secret_Bob_fast` in `SIDH.mag` in [16].)

Input: Alice's public key $\mathrm{PK}_{\mathrm{Alice}} = [a_A, x_{\phi_A(P_B)}, x_{\phi_A(Q_B)}, x_{\phi_A(Q_B - P_B)}]$, a list of positive integers L_B of length e_B describing the isogeny-graph strategy and Bob's secret key $\mathrm{SK}_{\mathrm{Bob}} = m_B \in \{1, 2, \ldots, \ell_B^{e_B - 1} - 1\} \cdot \ell_B$.

Output: A shared secret j-invariant of an elliptic curve $E_{\mathrm{shared}} : y^2 = x^3 + (A/C)x^2 + x$.

```
$A \leftarrow \mathrm{PK}_{\mathrm{Alice}}[1],\ C \leftarrow 1$
$X_R, Z_R \leftarrow$ LADDER_3_pt$(\mathrm{SK}_{\mathrm{Bob}}, \mathrm{PK}_{\mathrm{Alice}}[2], \mathrm{PK}_{\mathrm{Alice}}[3], \mathrm{PK}_{\mathrm{Alice}}[4], A)$
pts = [ ], $i = 0$
for $j = 1$ to $e_B$ do
    while $i < e_B - j$ do
        Push $(X_R, Z_R, i)$ to pts, i.e. append it to the end of pts
        $e \leftarrow L_B[e_B - i - j + 1]$
        $X_R, Z_R =$ xTPLe$(X_R, Z_R, A, C, e)$
        $i \leftarrow i + e$
    end while
    $A, C,$ consts $\leftarrow$ get_3_isog$(X_S, Z_S)$
    for $S = (X_S, Z_S, i_S) \in$ pts do
        $X_S, Z_S \leftarrow$ eval_3_isog(consts, $X_S, Z_S$)
    end for
    Pop $(X_S, Z_S, i_S)$ from pts, i.e. remove it from the end of pts
    $X_R \leftarrow X_S,\ Z_R \leftarrow Z_S,\ i \leftarrow i_S$
end for
$A, C,$ consts $\leftarrow$ get_3_isog$(X_R, Z_R)$
return j_inv$(A, C)$
```