eSIDH: the revenge of the SIDH

Daniel Cervantes-Vázquez Eduardo Ochoa-Jiménez
Francisco Rodríguez-Henríquez

January 7, 2020

Abstract

The Supersingular Isogeny-based Diffie-Hellman key exchange protocol (SIDH) was introduced by Jao an De Feo in 2011. SIDH operates on supersingular elliptic curves defined over $\mathbb{F}_{p^2}$, where p is a large prime number of the form $p = 4^{e_A}3^{e_B} - 1$, where e_A, e_B are positive integers such that $4^{e_A} \approx 3^{e_B}$. In this paper, a variant of the SIDH protocol that we dubbed extended SIDH (eSIDH) is presented. The eSIDH variant makes use of primes of the form, $p = 4^{e_A}\ell_B^{e_B}\ell_C^{e_C} f - 1$. Here ℓ_B, ℓ_C are two small prime numbers; f is a cofactor; and e_A, e_B and e_C are positive integers such that $4^{e_A} \approx \ell_B^{e_B}\ell_C^{e_C}$. We show that for many relevant instantiations of the SIDH protocol, this new family of primes enjoys a faster field arithmetic than the one associated to traditional SIDH primes. Furthermore, the proposed eSIDH protocol preserves the length and format of SIDH private/public keys, and its richer opportunities for parallelism yields a noticeable speedup factor when implemented on multi-core platforms. Using a single-core SIDH p_{751} implementation as a baseline, a parallel eSIDH p_{765} instantiation yields an acceleration factor of $1.05, 1.30$ and 1.41, when implemented on $k = \{1, 2, 3\}$-core processors.

1 Introduction

In 2011, Jao and De Feo proposed the Supersingular Isogeny-based Diffie-Hellman key exchange protocol (SIDH) [14] (see also [11]). Thanks to the high complexity of its underlying hard problem, SIDH provides key sizes comparable to classical public-key cryptosystems currently in use. Consequently, SIDH has been studied and implemented in an impressive number of recent publications [8, 10, 17, 13, 21, 6]. Moreover, the Supersingular Isogeny Key Encapsulation (SIKE) protocol [3], which can be seen as a descendant of SIDH, is one of the candidate schemes still under consideration within the second round of the NIST post-quantum cryptography standardization project [19].

The key exchange SIDH protocol operates on supersingular elliptic curves defined over $\mathbb{F}_{p^2}$, where p is a large prime number of the form $p = 4^{e_A}3^{e_B} - 1$. During the SIDH *Key Generation* and *Key Agrement* phases, Alice and Bob must compute degree-4^{e_A} and degree-3^{e_B} isogenies, respectively. Hence, by choosing the exponents e_A and e_B such that $4^{e_A} \approx 3^{e_B}$, one can assure that Alice and Bob will invest about the same computational expenses when executing SIDH. Moreover, this design choice also guarantees a healthy security balance due to the fact that the security guarantees of SIDH lie in the intractability of the Computational Supersingular Isogeny (CSSI) problem. Solving CSSI implies computing $\mathbb{F}_{p^2}$-rational isogenies of degrees 4^{e_A} and 3^{e_B} between pairs of supersingular elliptic curves defined over a quadratic extension field $\mathbb{F}_{p^2}$. It has been argued that the van Oorschot-Wiener golden collision finding algorithm is the most efficient classical or quantum attack on CSSI, having an expected running time of $O(p^{1/4})$ [1, 15, 9]. Moreover, if the bit-length of the integers 4^{e_A} and 3^{e_B} are highly unbalanced, Petit has shown in [20] that heuristic polynomial time key recovery attacks can be applied against SIDH.

In this paper, a variant of the SIDH protocol that allows us to accelerate Bob's computations on single and multi-core platforms without modifying the formats and lengths of its private/public keys is presented. The SIDH variant proposed in this paper is dubbed Extended-SIDH (eSIDH),[1] because of the pair of primes assigned to Bob for performing his isogeny computations. The eSIDH domain parameters are a supersingular elliptic curve $E/\mathbb{F}_{p^2}$, where p is a prime of the form,

$$p = 4^{e_A}\ell_B^{e_B}\ell_C^{e_C}f - 1. \tag{1}$$

Here ℓ_B, ℓ_C are two small prime numbers;[2] f is a cofactor; and e_A, e_B and e_C are positive integers such that $4^{e_A} \approx \ell_B^{e_B}\ell_C^{e_C}$.

Just as it would happen in the SIDH protocol, in the eSIDH instantiation Alice limits herself to compute degree-4^{e_A} isogenies. This naturally implies that Alice can still take advantage of the cheap cost associated to the fast degree-4 isogeny arithmetic. On the other hand, Bob is now responsible of computing degree-$\ell_B^{e_B}\ell_C^{e_C}$ isogenies. At first glance it would appear that Bob's task in eSIDH has just become more expensive than what used to be his computational role on a traditional SIDH scheme. Nonetheless, we will show in this paper that Bob's eSIDH tasks offer several advantages such as a faster underlying field arithmetic, and novel opportunities for exploiting the parallelism associated to his new computational responsibilities. Indeed, the rich abundance of the family of primes given in Eq. 1, produces for certain instantiations of eSIDH a faster field arithmetic by taking advantage of friendlier Montgomery-friendly primes [4, 10]. From our experimental results, we report that the computational advantages of eSIDH more than well compensate the extra computations demanded by this variant. For example, using a single-core SIKE prime p_{751} implementation as a baseline, a comparable eSIDH prime p_{765} instantiation yields an acceleration factor of $1.05, 1.30$ and 1.41, when implemented on $k = \{1, 2, 3\}$-core processors.

The remainder of this paper is organized as follows. In §2 a summary of the SIDH protocol and associated implementations aspects is presented. In §3 three different approaches for implementing the eSIDH protocol are presented. In §4 several relevant eSIDH implementations aspects on single-core and multi-core processors are discussed. We draw our concluding remarks in §5.

[1] Pronounced "e-side". An early version of this paper was presented in [2] and [18, Chapter11]

[2] In the eSIDH instantiations described in this paper we always choose $\ell_B = 3, \ell_C = 5$.

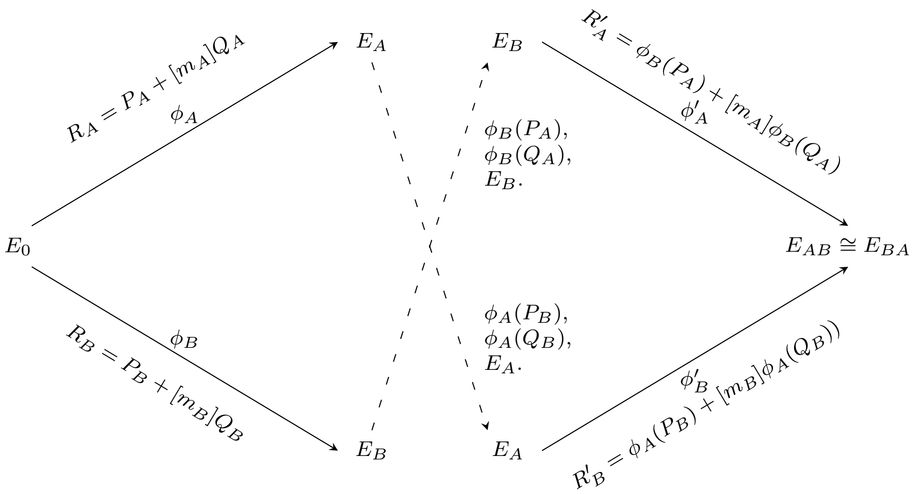

Figure 1: Overview of the SIDH protocol as proposed in [11]

2 Preliminaries

2.1 The SIDH protocol

The key exchange SIDH protocol operates on supersingular elliptic curves defined over $\mathbb{F}_{p^2}$, where p is a large prime number of the form $p = 4^{e_A}3^{e_B} - 1$. The exponents e_A and e_B are typically chosen such that $4^{e_A} \approx 3^{e_B}$. Let us define the constants $r_A = 4^{e_A}$ and $r_B = 3^{e_B}$. The public parameters of SIDH are given by a supersingular base curve E_0, and the basis points $P_A, Q_A, P_B, Q_B \in E_0$, such that $\langle P_A, Q_A \rangle = E_0[r_A]$ and $\langle P_B, Q_B \rangle = E_0[r_B]$. An overview of the SIDH protocol as it was proposed in [11] is depicted in Figure 1.

During the initial *Key Generation* phase, Alice chooses a random integer $m_A \in [1, r_A - 1]$, which acts as her secret key. Thereafter, Alice computes a secret key $R_A = P_A + [m_A]Q_A$ and a a degree-4^{e_A} isogeny public curve E_A such that $\phi_A : E_0 \to E_A$ with $\text{Ker}(\phi_A) = \langle R_A \rangle$. Likewise, Bob chooses a secret random integer $m_B \in [1, r_B - 1]$. Then, Bob computes a secret key $R_B = P_B + [m_B]Q_B$ and a degree-3^{e_B} isogeny public curve E_B such that $\phi_B : E_0 \to E_B$ with $\text{Ker}(\phi_B) = \langle R_B \rangle$. These computations complete the *Key Generation* phase.

During SIDH second phase, known as the *Key Agrement* phase, Alice sends Bob the tuple $[E_A, \phi_A(P_B), \phi_A(Q_B)]$, whereas Bob sends Alice the tuple $[E_B, \phi_B(P_A), \phi_B(Q_A)]$.[3] Alice uses Bob's information to recover the image of her secret key under Bob's curve E_B, as $\phi_B(R_A) = \phi_B(P_A) + [m_A]\phi_B(Q_A)$. Then Alice computes the curve E_{BA} such that there is a degree-4^{e_A} isogeny $\phi_{BA} : E_B \to E_{BA}$ with $\text{Ker}(\phi_{BA}) = \langle \phi_B(R_A) \rangle$. Similarly, Bob's

[3] State-of-the-art SIDH implementations use differential point arithmetic on Montgomery curves. Consequently, Alice and Bob evaluate and transmit three points each, namely, $x(P_A), x(Q_A), x(P_A - Q_A)$; and $x(P_B), x(Q_B)$, and $x(P_B - Q_B)$, respectively [8].

recovers the image of his secret key under Alice's curve E_A by computing $\phi_A(R_B) = \phi_A(P_B) + [m_B]\phi_A(Q_B)$. Bob then computes the isogenous curve E_{AB} such that there is a degree-3^{e_B} isogeny $\phi_{AB} : E_A \to E_{AB}$ with $\mathrm{Ker}(\phi_{AB}) = \langle \phi_A(R_B) \rangle$. This ends the SIDH protocol. Alice and Bob can now create a shared secret by computing the j-invariant of their respective curves, using the fact that $E_{BA} \cong E_{AB}$ implies $j(E_{BA}) = j(E_{AB})$.

Remark 1. The most prominent SIDH computational tasks include the computation of large degree isogenies and the evaluation of elliptic curve points in those isogenies. Another large operation of this scheme is the computation of four three-point scalar multiplications. For a typical software or hardware implementation of SIDH, the isogeny computations and associated point evaluations on one hand, along with the three-point scalar multiplications on the other hand, may take 70-80% and 20-30% of the overall protocol's computational cost, respectively.

Remark 2. In order to compute the points $R_A, \phi_B(R_A)$ (resp. $R_B, \phi_A(R_B)$), Alice (resp. Bob) must perform two three-point scalar multiplication procedures using a right-to-left Montgomery ladder algorithm [14, 10]. This kind of Montgomery ladder has a per-step cost of one point addition (xADD) and one point doubling (xDBL), which are usually performed in the projective space $\mathbb{P}^1$. Noticing that for current state-of-the-art SIDH implementations the costs of xDBL and xADD are about the same, one can assume that the per-step computational cost of the three-point Montgomery ladder is essentially that of two xDBL operations. It follows that the cost of computing $R_A, \phi_B(R_A)$ (resp. $R_B, \phi_A(R_B)$) is of $4e_A$ (resp. $2\log_2(3)e_B$) xDBL operations.

2.2 Optimal strategies for SIDH

Let E be a supersingular elliptic curve defined over the quadratic extension field $\mathbb{F}_{p^2}$. Given a point $R_0 \in E$, let $S = \langle R_0 \rangle$ be an order-ℓ^e subgroup of $E[\ell^e]$. Then there exists an isogeny $\phi : E \to E'$ (with both ϕ and E' defined over $\mathbb{F}_{p^2}$) having kernel S. The isogeny ϕ is unique up to isomorphism. Given E and S, an isogeny ϕ with kernel S and the corresponding equation for E', can be computed as a sequence of degree-ℓ isogenies using Vélu-like formulas and scalar multiplications by ℓ such as the ones discussed in [7, 5]. The optimal computation of large smooth-degree isogenies was presented and solved in [11].

In order to efficiently compute a degree-ℓ^e isogeny, it was shown in [11] that one can apply balanced or optimal strategies for traversing a weighted directed graph, which is represented in this paper as a right triangular lattice Δ_e having $\frac{e(e+1)}{2}$ points distributed in e columns and rows (See Figure 2a).[4] A leaf is defined as the most bottom point in a given column of the lattice. The vertexes of the graph represent elliptic curve points and its vertical and horizontal edges have as associated weight p_ℓ and q_ℓ, defined as the cost of performing one scalar multiplication by ℓ and one degree-ℓ isogeny, respectively. At the beginning of the isogeny computation, only the point R_0 of order ℓ^e is known. The goal of the isogeny construction/evaluation computation is to obtain one by one, all the

[4]Note that we depart from the tradition that would represent the weighted directed graph Δ_e as a triangular equilateral lattice between the x-axis and the lines $y = \sqrt{3}x$ and $y = -\sqrt{3}(x - e - 1)$.(cf. [11])

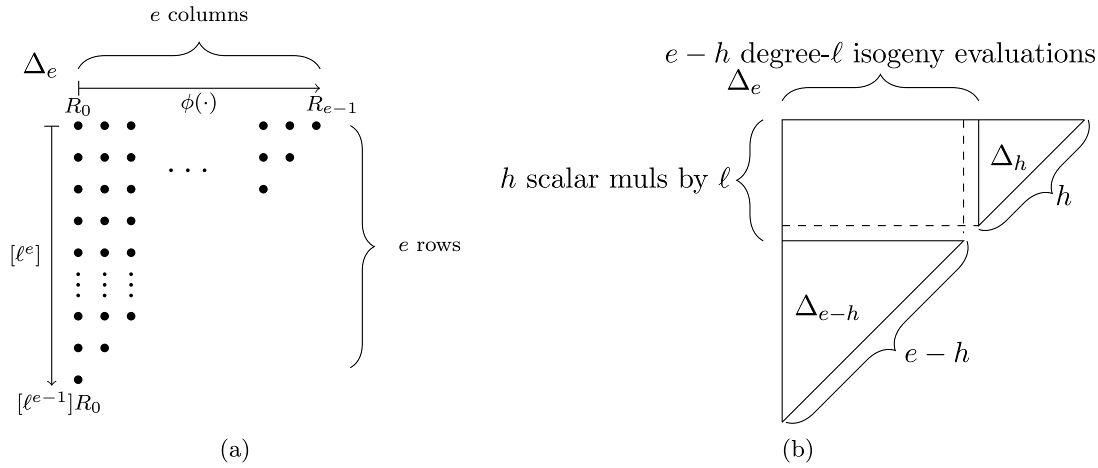

Figure 2: Subfigure 2a shows a triangular lattice used to compute a degree-ℓ^e isogeny $\phi : E \to E'$. The kernel of ϕ is the subgroup $\langle R_0 \rangle$, where $R_0 \in E$ is an order-ℓ^e elliptic curve point. Using an optimal SIDH strategy as in [11], a triangular lattice Δ_e is processed by splitting it into two sub-triangles as shown in Subfigure 2b. After applying this splitting strategy recursively, the cost of computing ϕ drops to approximately $\frac{e}{2}\log_2 e$ scalar multiplications by ℓ, $\frac{e}{2}\log_2 e$ degree-ℓ isogeny evaluations, and e constructions of degree-ℓ isogenous curves.

leaves in Δ_e until the farthest right one, R_{e-1}, has been calculated. Then, $\phi : E \to E'$ can be obtained by simply computing a degree-ℓ isogeny with kernel R_{e-1}.

Optimal strategies as defined in [11] exploit the fact that a triangle Δ_e can be optimally and recursively decomposed into two sub-triangles Δ_h and Δ_{e-h} as shown in Figure 2b. Let us denote as Δ_e^h the design decision of splitting a triangle Δ_e at row h. Then, the sequential cost of walking through the triangle Δ_e using the cut Δ_e^h is given as,

$$C(\Delta_e^h) = C(\Delta_h) + C(\Delta_{e-h}) + (e-h) \cdot q_\ell + h \cdot p_\ell.$$

We say that $\Delta_e^{\hat{h}}$ is optimal if $C(\Delta_e^{\hat{h}})$ is minimal among all Δ_e^h for $h \in [1, e-1]$. Applying this strategy recursively leads to a procedure that computes a degree-ℓ^e isogeny at a cost of approximately $\frac{e}{2}\log_2 e$ scalar multiplications by ℓ, $\frac{e}{2}\log_2 e$ degree-ℓ isogeny evaluations, and e constructions of degree-ℓ isogenous curves.

Remark 3. Let us assume that a degree-ℓ^e isogeny $\phi : E \to E'$ has been constructed using the procedure just described. Then given a point $P \in E$, its image $\phi(P) \in E'$ can be found by performing the composition of e degree-ℓ isogeny evaluations. As a way of illustration, the computation of the image of the point R_{BC} under Bob's isogeny $\phi_B(\cdot)$ is depicted in Figure 3 as the top horizontal segment of the triangular lattice going from the vertex R_{BC} to the vertex $\phi_B(R_{BC})$. The cost of this operation is of e_B degree-ℓ_B isogeny evaluations.

Protocol	Single Core processor required number of xDBL operations	Two-Core processor required number of xDBL operations
SIDH [11]	$\frac{16\lambda}{4}$	$\frac{16\lambda}{4}$
Naive §3.1	$\frac{16\lambda}{4}$	$\frac{16\lambda}{4}$
Parallel §3.2	$\frac{16\lambda}{4}$	$\frac{11\lambda}{4}$
CRT-based §3.3	$\frac{15\lambda}{4}$	$\frac{13\lambda}{4}$

Table 1: Let $\lambda = \lceil \log_2(p) \rceil$ be the bit-length of the eSIDH prime p. This table reports the approximate number of xDBL operations processed by the SIDH protocol of [11] compared against the three eSIDH variants discussed in this section (for the experimental clock cycle cost of xDBL see Table 3).

3 The extended SIDH (eSIDH) Protocol

The extended SIDH (eSIDH) Protocol operates on supersingular elliptic curves defined over $\mathbb{F}_{p^2}$, where p is a large prime number of the form $p = 4^{e_A} \ell_B^{e_B} \ell_C^{e_C} - 1$. The exponents e_A, e_B and e_C are chosen so that $4^{e_A} \approx \ell_B^{e_B} \ell_B^{e_C}$. The eSIDH protocol flow is quite similar to the one of a traditional SIDH as described in §2.1. Alice must still compute degree-4^{e_A} isogenies, but now Bob is responsible for computing degree-$\ell_B^{e_B} \ell_C^{e_C}$ isogenies.

In this section, three different approaches for computing the eSIDH protocol are presented. We start in §3.1 with the description of a simple naive eSIDH approach that is relatively expensive and offers little opportunities for exploiting parallelism. In §3.2, an eSIDH approach especially designed for exploiting parallelism opportunities is presented. Then, §3.3 presents a more economical eSIDH variant for single-core implementations, whose savings come from conveniently invoking the Chinese Remainder Theorem (CRT).

Table 1 shows the estimated scalar multiplication expenses incurred by SIDH and the three eSIDH instantiations discussed in this section. All the costs are given in number of xDBL operations. For single-core implementations, the CRT-based eSIDH protocol yields faster computational timings than the traditional SIDH protocol. In the case of two-core implementations, the parallel eSIDH described in §3.2, is significantly faster than the SIDH implementation of [3] and any other eSIDH instantiation discussed here.

3.1 A naive approach for computing eSIDH

Mimicking his role in SIDH, in a naive eSIDH instantiation Bob can first choose a basis for $\langle P_{BC}, Q_{BC} \rangle = E[\ell_B^{e_B} \cdot \ell_C^{e_C}]$. Thereafter, Bob computes his secret point as $R_{BC} = P_{BC} + [m_{BC}]Q_{BC}$ followed by the computation of a degree-$\ell_B^{e_B} \ell_C^{e_C}$ isogeny using an optimal strategy *à la* SIDH as shown in Figure 3.

Alice's eSIDH computational expenses are exactly the same as in SIDH. In the case of Bob, we stress that the computational expense of computing his eSIDH secret point R_{BC} as defined above, is about the same of computing Bob's SIDH secret point R_B as given in §2.1.

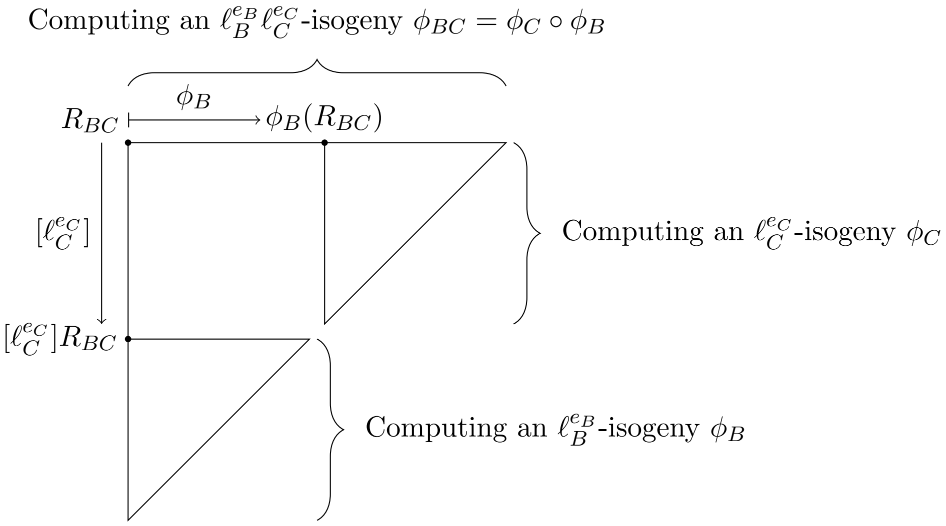

Figure 3: Overview of an strategy for computing a degree-$\ell_B^{e_B}\ell_C^{e_C}$ isogeny. Each isogeny ϕ_B and ϕ_C can be computed using a traditional SIDH strategy as in [11]. The kernel of ϕ_B is the subgroup $\langle[\ell_C^{e_C}]R_{BC}\rangle$, and the kernel of ϕ_C is the subgroup $\langle\phi_B(R_{BC})\rangle$.

Figure 3 depicts an optimal strategy procedure for computing Bob's degree-$\ell_B^{e_B}\ell_C^{e_C}$. The computational cost of this isogeny is of about $\frac{e_B}{2}\log_2 e_B$, $\frac{e_C}{2}\log_2 e_C$ scalar multiplications by ℓ_B and ℓ_C, $\frac{e_B}{2}\log_2 e_B$ degree-ℓ_B and $\frac{e_C}{2}\log_2 e_C$ degree-ℓ_C isogeny evaluations, and e_B and e_C constructions of degree-ℓ_B and degree-ℓ_C isogenous curves, respectively. This computational expense is nearly the same as the one required by Alice for computing a degree-4^{e_A} isogeny, using the optimal strategies described in §2.2 and Figure 2.

There seems to be no obvious way of parallelizing the main computation of this naive eSIDH instantiation. In the following two subsections, two eSIDH instantiations more amenable for parallelization are described.

3.2 A parallel approach for computing eSIDH

As mentioned before, eSIDH offers rich opportunities for exploiting its inherent parallelism. In this subsection an eSIDH instantiation specifically designed for the concurrent computation of this protocol's scalar multiplication operations will be presented.

As before, let $\lambda = \lceil\log_2(p)\rceil$ be the bit-length of the eSIDH prime $p = 4^{e_A}\ell_B^{e_B}\ell_C^{e_C} - 1$. For the sake of compactness let us define $r_B = \ell_B^{e_B}$ and $r_B = \ell_C^{e_C}$. Rather than defining Bob's secret point R_{BC} as in the previous subsection, Bob has now two secret points that he can calculate by choosing two pairs of bases such that $\langle P_B, Q_B\rangle = E[r_B]$ and $\langle P_C, Q_C\rangle = E[r_C]$. Afterwards, Bob randomly chooses two integers $m_B \in [1, r_B - 1]$ and $m_C \in [1, r_C - 1]$ to compute his secret points as,

$$R_B = P_B + [m_B]Q_B; \qquad R_C = P_C + [m_C]Q_C. \tag{2}$$

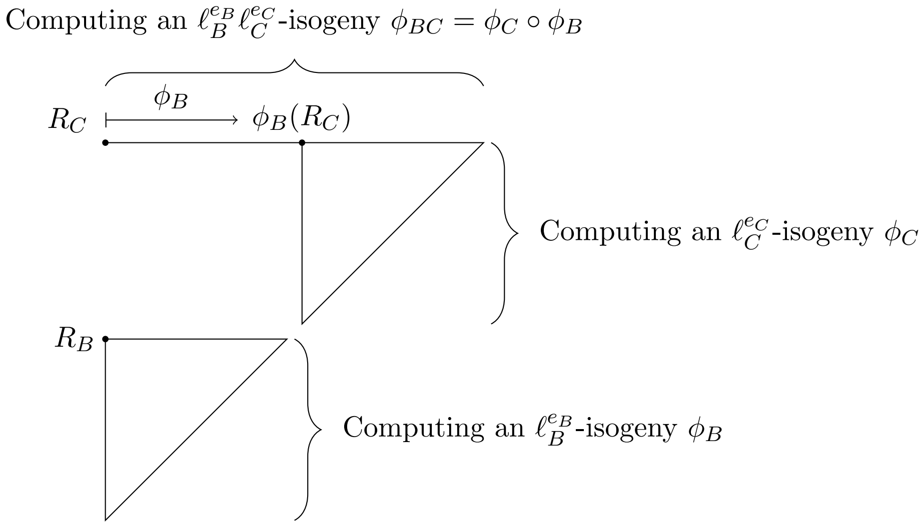

Figure 4: Overview of an strategy to compute an $\ell_B^{e_B}\ell_C^{e_C}$-isogeny, which exploits parallelism by defining two secret points R_B and R_C for Bob. Each isogeny ϕ_B and ϕ_C can be computed using a traditional SIDH strategy as in [11]. The kernel of ϕ_B is the subgroup $\langle R_B \rangle$, and the kernel of ϕ_C is the subgroup $\langle \phi_B(R_C) \rangle$.

Now, by picking ℓ_B, ℓ_C, e_B and e_C such that $\log_2(\ell_B)r_B \approx \log_2(\ell_C)r_C$, it follows that the cost of computing R_B is of about $\frac{2\lambda}{4}$ xDBL operations (cf. remark 2), which is nearly the same cost of computing R_C, and about half of the cost of computing Alice's secret point R_A. Furthermore, the calculations of Bob's secret points R_B and R_C are fully independent. Therefore, one can compute them in parallel on multi-core platforms. Moreover, the isogeny $\phi_{BC} = \phi_C \circ \phi_B$ can now be determined without performing the multiplication by r_C depicted in Figure 3. This computational saving comes from the facts that $gcd(r_B, r_C) = 1$ and that R_B, R_C are points of order r_B and r_C, respectively. Hence as shown in Figure 4, R_B and $\phi_B(R_C)$ can serve to generate the kernels of the isogenies ϕ_B and ϕ_C, respectively. This observation yields a significant saving of about $\frac{\lambda}{4}$ xDBL operations.

3.2.1 Reducing the public-key size of the parallel instantiation of eSIDH

Seemingly, an important drawback of using two secret points for Bob is that in the *Key Agrement* phase, this design decision forces Bob to know the images of his public points P_B, Q_B, P_C and Q_C, all of them evaluated under Alice's degree-4^{e_A} isogeny ϕ_A. Sending these four points implies an increment on the data to be transfered from Alice to Bob. This in turn implies an increment on Alice's computational load since now, she would need to find the isogeny images of four points (instead of two as in the original SIDH).[5]

[5] In practice one uses differential point arithmetic on Montgomery curves. Hence, Alice would need to evaluate and transmit six points, namely, $x(P_B), x(Q_B), x(P_B - Q_B), x(P_C), x(Q_C)$, and $x(P_B - Q_B)$.

Alternatively, one can reduce the eSIDH public-key size at the same time that Alice's extra work is prevented. This can be done by defining two auxiliary public points that while codifying Bob's public points P_B, Q_B, P_C and Q_C, provide an efficient way to recover them. Let us re-define Bob's public points as $S = P_B + P_C$ and $T = Q_B + Q_C$. This implies that,

$$[r_B]S = [r_B]P_C, \quad [r_C]S = [r_C]P_B, \quad [r_B]T = [r_B]Q_C, \quad \text{and} \quad [r_C]T = [r_C]Q_B. \tag{3}$$

Hence, given the points S, T, one can recover multiples of Bob's original four public points by performing four scalar multiplications. Notice that all four of these scalar multiplications are fully independent. Nonetheless, we can do better as discussed below.

Remark 4. From the multiples $[r_C]P_B$ and $[r_C]Q_B$, one can recover the points P_B, Q_B, by multiplying them by the scalars $r_C^{-1} \bmod r_B$ and $r_B^{-1} \bmod r_C$, respectively. However, it is easier to directly use $[r_C]P_B$ and $[r_C]Q_B$ to generate the point $R'_B = [r_C]P_B + [m_B]([r_C]Q_B)$. Provided that $gcd(r_C, r_B) = 1$, it follows that $R'_B = [r_C]R_B$. Thus, $\langle R'_B \rangle = \langle R_B \rangle$, which implies that the degree-r_C isogenies with kernels $\langle R'_B \rangle$ and $\langle R_B \rangle$, are the same up to isomorphism. Similarly, the point $R'_C = [r_B]P_C + [m_C]([r_B]Q_C)$, is sufficient to generate the degree-r_B isogeny with kernel $\langle R_C \rangle$ up to isomorphism.

The observation stated in Remark 4 along with the relations given in Eq. (3) suggest an approach where Bob can efficiently recover the points R'_B, R'_C, by the direct computation of,

$$R'_B = [r_C](S + [m_B]T) \qquad \text{and} \qquad R'_C = [r_B](S + [m_C]T). \tag{4}$$

Remark 5. Eq. (4) is useful during the eSIDH *Key Agrement* phase. For the eSIDH *Key Generation* phase, it becomes more efficient to compute the points R_B and R_C as discussed at the beginning of Subsection 3.2.

Figure 5 shows a general overview of the eSIDH parallel instantiation described in this subsection. Assuming that a multi-core platform is available for the execution of this eSIDH instantiation, most Bob's scalar multiplications can be computed in parallel.

Remark 6. **eSIDH security**: Recall that $gcd(r_B, r_C) = 1$ and $r_A \approx \log_2(\ell_B)r_B \cdot \log_2(\ell_C)r_C$. Given the points S and T, computing a degree-$r_B r_C$ isogeny between E_0 and E_{BC} should have the same computational complexity as the problem of, given the points P_A and Q_A, finding a degree-r_A isogeny between E_0 and E_A. Furthermore, provided that $4^{e_A} \approx \ell_B^{e_B} \cdot \ell_C^{e_C}$, the heuristic polynomial time key recovery attacks presented in [20] do not appear to apply against eSIDH.

3.2.2 Computational cost of the eSIDH parallel instantiation

As in Table 1, the eSIDH required number of xDBL operations will be used as cost metric. We further assume that $\log_2(\ell_B)r_B \approx \log_2(\ell_C)r_C \approx \frac{r_A}{2} \approx \frac{\lambda}{4}$.

Note that the private/public key sizes of eSIDH are the same as the traditional SIDH protocol of [11]. Moreover, Alice's isogeny computations are exactly the same for both protocols. Nevertheless, Bob can compute his two degree-$\ell_B^{e_B}\ell_C^{e_C}$ isogenies using

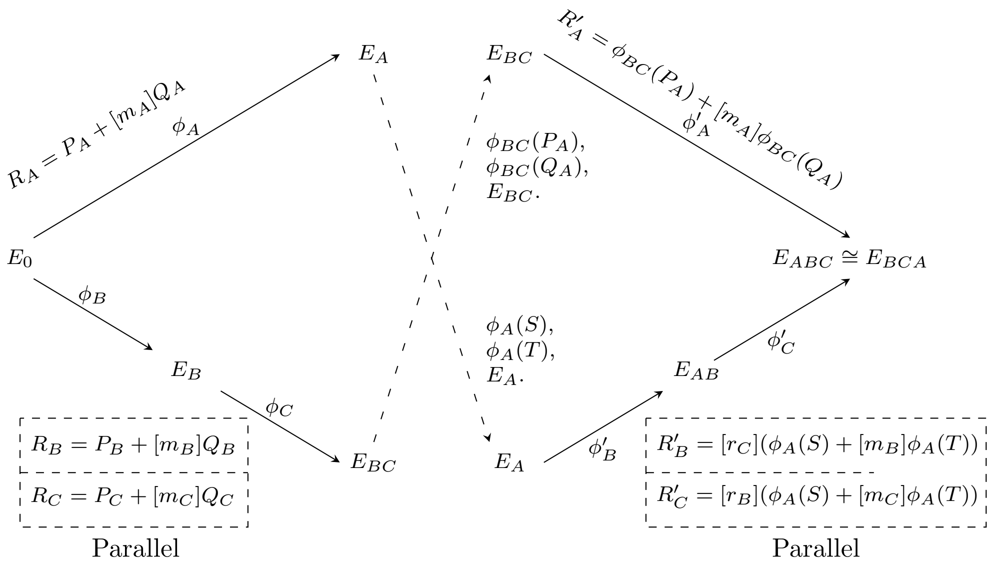

Figure 5: Overview of an eSIDH parallel instantiation with Bob's secret points computed in parallel. In the *Key Generation* phase $\text{Ker}(\phi_B) = \langle R_B \rangle$ and $\text{Ker}(\phi_C) = \langle \phi_B(R_C) \rangle$. In the *Key Agrement* phase $\text{Ker}(\phi'_B) = \langle R'_B \rangle$ and $\text{Ker}(\phi'_C) = \langle \phi'_B(R'_C) \rangle$

the computational trick shown in Figure 4. This approach yields a saving of about $\frac{2\lambda}{4}$ xDBL operations compared against the computational cost required by the SIDH strategy shown in Figure 2.

The scalar multiplications computational expenses of the parallel eSIDH variant are dispensed as discussed next. Let us consider the eSIDH instantiation depicted in Figure 5. Then, as in the traditional SIDH, Alice must perform two $\frac{2\lambda}{4}$-bit scalar multiplications that involve the computation of about $\frac{8\lambda}{4}$ xDBL operations (cf. Remark 2). Moreover, during the *Key Generation* phase, Bob computes the points R_B and R_C, by performing $\frac{4\lambda}{4}$ and $\frac{2\lambda}{4}$ xDBL operations for a single-core and two-core implementation, respectively. During the *Key Agrement* phase, Bob computes the points R'_B, R'_C, by performing $\frac{6\lambda}{4}$ and $\frac{3\lambda}{4}$ xDBL operations for a single-core and two-core implementation, respectively.

Thus, the eSIDH combined scalar multiplication effort of Alice and Bob for a a single-core and two-core implementation is of $\frac{16\lambda}{4}$ and $\frac{11\lambda}{4}$, respectively (see Table 1).

3.3 A CRT-based approach for computing eSIDH

Another instantiation of eSIDH can be constructed by taking advantage of the Chinese Remainder Theorem (CRT). As in §3.2, let $\lambda = \lceil \log_2(p) \rceil$ be the bit-length of the eSIDH prime $p = 4^{e_A} \ell_B^{e_B} \ell_C^{e_C} - 1$. For the sake of compactness let us define $r_B = \ell_B^{e_B}$ and $r_B = \ell_C^{e_C}$. A CRT-based approach for eSIDH can be computed as explained in the remainder of this subsection.

First choose a pair of random integers under the following restrictions. Pick randomly $m_B \in [1, r_B]$ and $m_C \in [1, r_C]$ such that, $gcd(m_B, r_C) = gcd(m_C, r_B) = 1$. Then compute the following integers,

$$\begin{aligned} \hat{m}_B &= m_B^{-1} \bmod r_C; & \hat{m}_C &= m_C^{-1} \bmod r_B; \\ \bar{m}_B &= m_B \cdot \hat{m}_B \bmod r_B; & \bar{m}_C &= m_C \cdot \hat{m}_C \bmod r_C; \\ m_{BC} &= m_B \cdot \hat{m}_B \cdot m_C \cdot \hat{m}_C \bmod (r_B \cdot r_C). \end{aligned} \tag{5}$$

From Eq. (5) it follows that $m_{BC} \equiv \bar{m}_B \mod r_B$ and $m_{BC} \equiv \bar{m}_C \mod r_C$.

For the execution of the eSIDH *Key Generation* phase the following two points are computed, $R_B = P_B + [\bar{m}_B]Q_B$ and $R_C = P_C + [\bar{m}_C]Q_C$. Thereafter, one can compute ϕ_{BC} as shown in Figure 4, such that the kernel of ϕ_B is generated by R_B and the kernel of ϕ_C is generated by $\phi_B(R_C)$. Since $|m_B| \approx |m_C| \approx \frac{|m_A|}{2} = \frac{\lambda}{4}$,[6] the combined cost of computing R_B and R_C is about the same as the cost of computing R_A. As a side effect, note that these computations imply a saving of $r_C \approx \frac{\lambda}{4}$ xDBL operations corresponding to the left most vertical edge between the points R_C and R_B shown in Figure 4.

For the computation of the eSIDH *Key Agrement* phase as in §3.2.1, let us define the auxiliary public points $S = P_B + P_C$ and $T = Q_B + Q_C$. It turns out that the generators of the subgroups $\langle R_B \rangle$ and $\langle R_C \rangle$ can be recovered by invoking the CRT and Remark 4 applied on the integers given in Eq. (5).

Proposition 1. *Let P_B, Q_B, P_C, Q_C, $\bar{m}_B$, $\bar{m}_C$, m_{BC}, R_B as R_C be defined as before, and fix $S = P_B + P_C$ and $T = Q_B + Q_C$. Then $[r_C]R_B = [r_C](S + [m_{BC}]T)$ and $[r_B]R_C = [r_B](S + [m_{BC}]T)$.*

Proof. By straightforward substitution we get,

$$\begin{aligned} [r_C](S + [m_{BC}]T) =& [r_C](P_B + P_C + [m_{BC}]Q_B + [m_{BC}]Q_C)) \\ =& [r_C](P_B + [m_{BC}]Q_B) \\ =& [r_C](P_B + [m_{BC} \mod r_B]Q_B) \\ =& [r_C](P_B + [\bar{m}_B]Q_B) \\ =& [r_C]R_B. \end{aligned}$$

Using an analogous procedure one can show that $[r_B]R_C = [r_B](S + [m_{BC}]T)$. □

Using Proposition 1, one can recover the generator R'_B of the subgroup $\mathrm{Ker}(\phi'_B)$ and $\phi'_B(R'_C)$, the generator of the subgroup $\mathrm{Ker}(\phi'_C)$. To this end, one can compute,

$$\begin{aligned} R'_B &= [r_C](\phi_A(S) + [m_{BC}]\phi_A(T)) = \phi_A([r_C]R_B); \\ R'_C &= [r_B](\phi_A(S) + [m_{BC}]\phi(T)) \quad = \phi_A([r_B]R_C). \end{aligned}$$

Nevertheless, these computations have a steep cost of $\frac{10\lambda}{4}$ xDBL operations. Fortunately, there is an efficient way to reduce this expense.

[6] The operator $|\cdot|$ evaluates the bit-length of its operand.

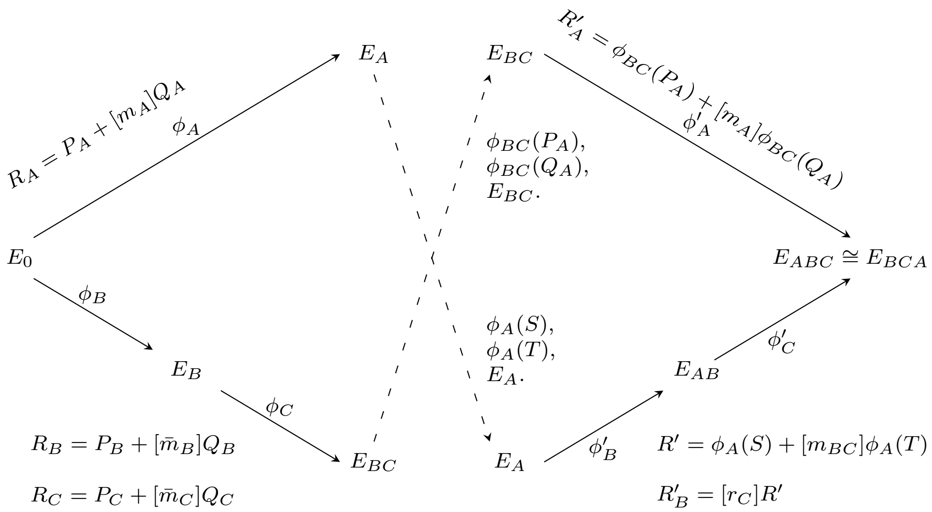

Figure 6: Overview of the CRT-based eSIDH instantiation. In the *Key Generation* phase $\text{Ker}(\phi_B) = \langle R_B \rangle$ and $\text{Ker}(\phi_C) = \langle \phi_B(R_C) \rangle$. In the *Key Agrement* phase $\text{Ker}(\phi'_B) = \langle R'_B \rangle$ and $\text{Ker}(\phi'_C) = \langle \phi'_B(R') \rangle$

Proposition 2. *Fix* $R'_B = [r_C](\phi_A(S) + [m_{BC}]\phi_A(T)) = [r_C]R'$. *The point* $\phi'_B(R')$ *has order* r_C *and* $\phi'_B(R') = \phi'_B((\phi_A(R_C))$.

Proof. By virtue of Proposition 1, the order-r_B point R_B generates the kernel of the degree-r_B isogeny ϕ'_B, that is, $\text{Ker}(\phi'_B) = \langle R'_B \rangle$. By straightforward substitution we get,

$$\begin{aligned} R' &= \phi_A(S + [m_{BC}]T) \\ &= \phi_A(P_B + [m_{BC}]Q_B + P_C + [m_{BC}]Q_C) \\ &= \phi_A(R_B + R_C). \end{aligned}$$

It follows that

$$\phi'_B(R') = \phi'_B((\phi_A(R_B + R_C)) = \phi'_B((\phi_A(R_B) + \phi_A(R_C)) = \phi'_B((\phi_A(R_C)),$$

which yields an order-r_C point. □

Note that the points R'_B and $\phi'_B(R')$ can serve as the kernel generators of Bob's key-agreement phase isogenies ϕ'_B and ϕ'_C, respectively. Moreover, the cost of computing those two points is of about $\frac{5\lambda}{4}$ xDBL operations. There seems to be no obvious way to parallelize these two calculations. Figure 6 depicts how this CRT-based variant of eSIDH can be computed.

3.3.1 Computational cost of the CRT-based eSIDH instantiation

As in §3.2.2, the eSIDH required number of xDBL operations will be used as cost metric, and we will assume that $\log_2(\ell_B)r_B \approx \log_2(\ell_C)r_C \approx \frac{r_A}{2} \approx \frac{\lambda}{4}$. Also, as argued in §3.2.2, the private/public key sizes of eSIDH and Alice's isogeny computations are exactly the same as in SIDH. Bob can compute his two degree-$\ell_B^{e_B}\ell_C^{e_C}$ isogenies using the computational trick shown in Figure 4, obtaining a saving of about $\frac{2\lambda}{4}$ xDBL operations compared against SIDH.

The scalar multiplications computational expenses of the CRT-based eSIDH variant are dispensed as discussed next. Let us consider the eSIDH instantiation depicted in Figure 5. Then, as in the traditional SIDH, Alice must perform two $\frac{2\lambda}{4}$-bit scalar multiplications that involve the computation of about $\frac{8\lambda}{4}$ xDBL operations (cf. Remark 2).

During the *Key Generation* phase, Bob computes the points R_B, R_C, by performing $\frac{4\lambda}{4}$ and $\frac{2\lambda}{4}$ xDBL operations for a single-core and two-core implementation, respectively. During the *Key Agrement* phase, Bob computes the points R', R'_B, by performing $\frac{5\lambda}{4}$ xDBL operations for either a single-core or a two-core implementation.

Thus, the eSIDH combined scalar multiplication effort of Alice and Bob for a single-core and a two-core implementation is of $\frac{15\lambda}{4}$ and $\frac{13\lambda}{4}$, respectively (see Table 1).

4 Parameter selection and implementation aspects

4.1 The hunting for efficient eSIDH Primes

Let $N = \lceil\lceil\log_2(p)\rceil/w\rceil$ be the minimum number of 64-bit words needed to represent an eSIDH prime p. In this paper it is assumed $w = 64$. We say that a modulus p is γ-Montgomery-friendly if $p \equiv \pm 1 \mod 2^{\gamma \cdot w}$ for a positive integer γ [12, 16]. This property implies that $-p^{-1} \equiv \mp 1 \mod 2^{\gamma \cdot w}$, which is conveniently exploited to produce savings in the Montgomery's REDC reduction algorithm [4].

The most common choice of SIKE primes is to use primes of the form $p := 4^{e_A}3^{e_B} - 1$. There are at least two computer arithmetic reasons for this choice. One of them, is that this family of primes are Montgomery-friendly, which implies that they admit fast Montgomery Reduction [10, 4]. The second advantage is that there exist highly efficient formulas for computing degree-3 and degree-4 isogenies [7, 5]. The eSIDH primes proposed in this paper are of the form $p := 4^{e_A}\ell_B^{e_B}\ell_C^{e_C}f - 1$, which are much more flexible and abundant than the SIKE primes. Then, given some fixed values for N and the primes ℓ_B and ℓ_C, one searches for $\frac{N}{2}$-Montgomery-friendly primes (if they exist) by varying e_B, e_C and f. These friendlier Montgomery-friendly primes achieve a faster Montgomery reduction (see [10, Algorithm 6]) than the one that could possibly be obtained from comparable SIKE primes.

Another important design aspect to be considered is that on Bob's side, there exists a trade-off between the size of the base-primes ℓ_B and ℓ_C and their corresponding exponents e_B and e_C, respectively. The base-primes define the *size of the step*, whereas their exponents determine how many steps one must perform for isogeny evaluations and constructions. Depending on the exact choice of these parameters, one can make a few big steps or many small steps. Furthermore as discussed in §3.2, in order to take full advantage of parallel computing and also for security reasons (cf. Remark 6), it is important to choose $\log_2(\ell_B)r_B \approx \log_2(\ell_C)r_C$.

eSIDH primes proposed here	N	γ	**SIKE primes as in [3]**	N	γ
$p_{443} = 2^{222}3^{73}5^{45} - 1$	7	3	$p_{434} = 2^{216}3^{137} - 1$	7	3
$p_{508} = 2^{258}3^{74}5^{57} - 1$	8	4	$p_{503} = 2^{250}3^{159} - 1$	8	3
$p_{628} = 2^{320}3^{94}5^{67} - 1$	10	5	$p_{610} = 2^{305}3^{192} - 1$	10	4
$p_{765} = 2^{391}3^{119}5^{81} - 1$	12	6	$p_{751} = 2^{372}3^{239} - 1$	12	5

Table 2: Our selection of eSIDH primes matching the four security levels offered by the SIKE primes included in [3], where $N = \lceil \lceil \log_2(p) \rceil / 64 \rceil$, and γ is the largest integer for that N such that $p \equiv -1 \mod 2^{\gamma \cdot 64}$ holds.

Operation	P_{434}	P_{443}	P_{751}	P_{765}
Reduction $\mathbb{F}_p$	78	78	154	137
Mult $\mathbb{F}_{p^2}$	466	467	1,029	977
Sqr $\mathbb{F}_{p^2}$	349	349	780	716
Inv $\mathbb{F}_{p^2}$	77,764	80,253	317,655	251,366
Doubling	2,961	2,920	6,186	5,845
4-IsoGen	1,793	1,758	3,691	3,442
4-IsoEval	3,955	3,921	8,407	7,972
Tripling	5,595	5,487	11,999	11,292
3-IsoGen	2,850	2,836	5,720	5,418
3-IsoEval	2,717	2,717	5,944	5,612
Quintupling	-	7,995	-	16,285
5-IsoGen	-	7,951	-	16,179
5-IsoEval	-	4,703	-	9,682

Table 3: Timing performance of selected quadratic field arithmetic operations and isogeny evaluations and constructions. Timings are reported in clock cycles measured on a Skylake processor at 4.0GHz.

For all the eSIDH instances considered in this paper, we use primes of the form $p = 4^{e_A}\ell_B^{e_B}\ell_C^{e_C}f - 1$, such that $2e_A \approx \log_2(\ell_B^{e_B}\ell_C^{e_C})$, and where e_A is chosen so that the security level offered by the SIKE primes as specified in [3] is matched (see also [1]). The cofactor $f = 2^k c$ is carefully selected so that p qualifies as an $\frac{N}{2}$-Montgomery-friendly prime (if at all possible). Table 2 shows our selection of four eSIDH primes matching the four security levels specified in [3]. When searching for eSIDH primes with comparable security as the one offered by the p_{434} SIDH prime, the best choice that we were able to find is p_{443} as specified in Table 2. Both of them, p_{434} and p_{443}, fit in seven 64-bit words and they are 3-Montgomery-friendly primes. This implies that the field arithmetic costs associated to p_{434} and p_{443} are fairly similar (cf. Table 3). Luckily, for the other three security levels we managed to find eSIDH $\frac{N}{2}$-Montgomery-friendly primes sharing the same security level as their SIKE prime counterparts.

Phase	p_{434}			p_{443}		
	Cores number			Cores number		
	1	2	3	1	2	3
Alice Key Generation	5.93	5.62	5.36	5.91	5.60	5.36
Bob Key Generation	6.54	6.20	5.88	6.53	5.03	4.54
Alice Key Agrement	4.80	4.49	4.22	4.78	4.48	4.22
Bob Key Agrement	5.50	5.14	4.82	6.23	4.88	4.55
Total	22.77	21.45	20.28	23.45	19.99	18.67

Table 4: Performance comparison of the SIKE prime p_{434} against the eSIDH prime p_{443}. All timings are reported in 10^6 clock cycles measured on an Intel Skylake proccessor at 4.0 GHz.

4.2 Results and discussion

In this subsection, a full implementation of the eSIDH protocol proposed in this work is presented. We mainly focus ourselves on the eSIDH parallel instantiation discussed in §3.2, and we use the SIDH implementation of [3] as a baseline to compare the acceleration factor achieved by the eSIDH scheme. Our two case studies targeted p_{434} and p_{751}, the smallest and largest SIKE primes that are included in the SIKE specification [3].

All the timings were measured using an Intel core i7-6700K processor with micro-architecture Skylake at 4.0 GHz. Using the Clang-3.9 compiler and the flags `-Ofast -fwrapv -fomit-frame-pointer -march=native -madx -mbmi2`.

4.2.1 Quadratic field arithmetic and isogeny computations

Table 3 presents a comparison of the field arithmetic costs associated to the SIKE primes p_{434} and p_{751} against the ones exhibit by the eSIDH primes p_{443} and p_{765}, respectively. Note that our eSIDH prime p_{765} field arithmetic gets noticeable timing speedups compared against the SIKE p_{751} field arithmetic. This acceleration is justified from the fact that the modular reduction of the former is faster than the latter.

4.2.2 Parallelizing the SIDH protocol

As of today, relatively few works have attempted to exploit the rich opportunities that the SIDH main computations can offer for parallel computations. In this direction, we are only aware of the works reported in [17, 13], where explicit efforts for parallelizing the computations of the SIDH protocol were attempted and/or exploited. Using the same approach followed in [17], in this work we parallelize the SIDH implementation of [3] as follows. Alice and Bob isogeny evaluations and constructions were computed using the optimal strategy of [11]. Optimal strategies typically produce an average of four points per curve whose isogeny images can be processed concurrently [17]. Hence, using a two- and three-core processor we actively strove for concurrently performing as many isogeny evaluations as possible.[7]

[7]Parallel canonical strategies for SIDH are studied and proposed in [13].

Phase	p_{751}			p_{765}		
	Cores number			Cores number		
	1	2	3	1	2	3
Alice Key Generation	23.59	21.74	19.88	22.27	20.19	18.89
Bob Key Generation	26.74	23.71	22.24	24.34	17.76	15.79
Alice Key Agrement	19.37	17.49	15.64	18.21	16.12	14.83
Bob Key Agrement	22.76	19.74	18.25	23.24	17.16	15.94
Total	92.46	82.67	76.01	88.05	71.23	65.42

Table 5: Performance comparison of the SIKE prime p_{751} against the eSIDH prime p_{765}. All timings are reported in 10^6 clock cycles measured on an Intel Skylake proccessor at 4.0 GHz.

Table 4 shows that with respect to a sequential implementation, a two-core and a three-core parallel implementation of the SIDH p_{434} instantiation yields a speedup factor of 1.062 and 1.123, respectively. Likewise, Table 5 reports that with respect to a sequential implementation, a two-core and a three-core parallel implementation of the SIDH p_{751} instantiation yields a speedup factor of 1.118 and 1.216, respectively.

4.2.3 Performance evaluation of the eSIDH parallel instantiation

Table 4 reports the performance timing achieved by the eSIDH p_{443} parallel instantiation. Using a single-core SIDH p_{434} implementation as a baseline, it can be seen from Table 4 that a parallel eSIDH p_{443} implementation yields an acceleration factor of $0.97, 1.22$ and 1.41, when executed on $k = \{1, 2, 3\}$-core processors.

On the other hand, eSIDH p_{443} yields an acceleration factor of $0.971, 1.073$ and 1.086, when both protocols are implemented on $k = \{1, 2, 3\}$-core processors. Hence for a single-core implementation, eSIDH p_{443} is slower than its SIDH p_{434} counterpart. For two-core and three-core implementations, our eSIDH variant produces a modest but noticeable speedup of about 7% and 9%, respectively.

Table 5 reports the performance timing achieved by the eSIDH p_{765} parallel instantiation. Using a single-core SIDH p_{751} implementation as a baseline, it can be seen that a parallel implementation of eSIDH p_{765} yields an acceleration factor of $1.05, 1.30$ and 1.41, when executed on $k = \{1, 2, 3\}$-core processors. Furthermore, eSIDH p_{765} yields an acceleration factor of $1.050, 1.160$ and 1.162. when both protocols are implemented on $k = \{1, 2, 3\}$-core processors. We stress that even for a single-core implementation of this case study, our eSIDH variant produces a modest but noticeable speedup of about 5%.

As a general summary we note that for single-core implementations, Bob's $3^{e_3}5^{e_5}$ isogeny computation has no overhead impact on the *Key Generation* phase. However, the public key recovery mechanism (cf. §3.2.1) proves to be relatively expensive on the *Key Agrement* phase. On the other hand, for two- and three-core implementations, our eSIDH instantiation clearly outperforms SIDH on all Bob's computations. In Table 5, the comparison of SIDH p_{751} against eSIDH p_{765} reveals the superiority of the latter over the former in all the phases of the protocol, even for a sequential implementation. The one exception being Bob's *Key Agrement* phase. For the two- and three- core implementations, Bob's *Key Agrement* for eSIDH p_{765} is even faster than Alice's *Key Agrement* for SIDH p_{751}.

4.2.4 Performance evaluation of the CRT-based eSIDH instantiation

A shown in Table 1, the CRT-based eSIDH instantiation presented in §3.3 offers less parallelism opportunities than the ones enjoyed by the eSIDH parallel instantiation discussed in 3.2. However, according to the estimates given in Table 1, the CRT-based eSIDH instantiation is a promising economical scheme for a sequential single-core processor. As before, let $\lambda = \lceil \log_2(p) \rceil$. Referring to Table 1, the computational cost of the CRT-based eSIDH instantiation saves $\approx \frac{\lambda}{4}$ xDBL operations.

Case study p_{443}
Based on the timing computational costs reported in Table 3, the expected computational saving for a single-core implementation of SIDH p_{434} with respect to eSIDH p_{443} is given as,

$$\begin{aligned} e_C \cdot \text{Quintupling} &= 45 \cdot 7995 \\ &= 359,775 \text{ clock cycles.} \end{aligned}$$

This implies that compared against a single-core SIDH p_{434} implementation, a single-core CRT-based eSIDH p_{443} implementation produces a 1.02 speedup factor.
Case study p_{765}
Based on the prime specifications given in 2 and the timing computational costs reported in Table 3, the expected computational saving for a single-core implementation of eSIDH p_{765} with respect to SIDH p_{751} is given as,

$$\begin{aligned} e_C \cdot \text{Quintupling} &= 81 \cdot 16285 \\ &= 1,319,085 \text{ clock cycles.} \end{aligned}$$

This saving combined with the experimental results reported in Table 5 implies that compared against a single-core SIDH p_{751} implementation, a single-core CRT-based eSIDH p_{765} implementation produces a 1.07 speedup factor.

5 Conclusions

In this paper the extended SIDH scheme, a variant of the SIDH protocol in [11], was presented. Our experimental results show that an eSIDH parallel implementation is faster than parallel version of the SIDH protocol. Furthermore for certain security levels, a CRT-based eSIDH single-core implementation is slightly faster than SIDH.

Our future work includes to expand the search of more efficient eSIDH primes for all the four security levels considered in [3]. Building on the work presented in [13], we would also like to explore efficient approaches for parallelizing the SIDH isogeny computations and evaluations. The algorithmic ideas discussed here might be useful for the B-SIDH construction [6], where given the large size of the prime factors involved in the factorization of $p \pm 1$, parallel implementations of SIDH become mandatory. We also would like to explore applications of eSIDH to the client-server scenarios discussed in [6].

Acknowledgements

This work was done while the first and third authors were visiting the University of Waterloo. The authors would like to thank Jesús-Javier Chi-Domínguez for his useful comments.

References

[1] G. Adj, D. Cervantes-Vázquez, J.-J. Chi-Domínguez, A. Menezes, and F. Rodríguez-Henríquez. On the cost of computing isogenies between supersingular elliptic curves. In C. Cid and M. J. Jacobson Jr., editors, *Selected Areas in Cryptography – SAC 2018*, pages 322–343. Springer International Publishing, 2019.

[2] D. C.-V. Eduardo Ochoa-Jiménez Francisco Rodríguez-Henríquez. A [magical] parallel variant of SIDH. CHES 2018 Rump session, 2018. `https://eprint.iacr.org/2019/1145`.

[3] R. Azarderakhsh, M. Campagna, C. Costello, L. D. Feo, B. Hess, A. Jalali, D. Jao, B. Koziel, B. LaMacchia, P. Longa, M. Naehrig, G. Pereira, J. Renes, V. Soukharev, and D. Urbanik. Supersingular isogeny key encapsulation. second round candidate of the NIST's post-quantum cryptography standardization process, 2017. Available at: https://sike.org/.

[4] J. W. Bos and S. Friedberger. Arithmetic considerations for isogeny-based cryptography. *IEEE Trans. Computers*, 68(7):979–990, 2019.

[5] D. Cervantes-Vázquez and F. Rodríguez-Henríquez. A note on the cost of computing odd degree isogenies. Cryptology ePrint Archive, Report 2019/1373, 2019. `https://eprint.iacr.org/2019/1373`.

[6] C. Costello. B-SIDH: supersingular isogeny Diffie-Hellman using twisted torsion. Cryptology ePrint Archive, Report 2019/1145, 2019. `https://eprint.iacr.org/2019/1145`.

[7] C. Costello and H. Hisil. A simple and compact algorithm for SIDH with arbitrary degree isogenies. In T. Takagi and T. Peyrin, editors, *Advances in Cryptology - ASIACRYPT 2017 - 23rd International Conference on the Theory and Applications*

of Cryptology and Information Security Part II, volume 10625 of *Lecture Notes in Computer Science*, pages 303–329. Springer, 2017.

[8] C. Costello, P. Longa, and M. Naehrig. Efficient algorithms for supersingular isogeny Diffie-Hellman. In M. Robshaw and J. Katz, editors, *Advances in Cryptology - CRYPTO 2016 - Part I*, volume 9814 of *Lecture Notes in Computer Science*, pages 572–601. Springer, 2016.

[9] C. Costello, P. Longa, M. Naehrig, J. Renes, and F. Virdia. Improved classical cryptanalysis of the computational supersingular isogeny problem. Cryptology ePrint Archive, Report 2019/298, 2019. `https://eprint.iacr.org/2019/298`.

[10] A. Faz-Hernández, J. López, E. Ochoa-Jiménez, and F. Rodríguez-Henríquez. A faster software implementation of the supersingular isogeny Diffie-Hellman key exchange protocol. *IEEE Transactions on Computers*, pages 1–1, 2018.

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

[12] S. Gueron and V. Krasnov. Fast prime field elliptic-curve cryptography with 256-bit primes. *Journal of Cryptographic Engineering*, pages 1–11, 2014.

[13] A. Hutchinson and K. Karabina. Constructing canonical strategies for parallel implementation of isogeny based cryptography. In D. Chakraborty and T. Iwata, editors, *Progress in Cryptology – INDOCRYPT 2018*, pages 169–189. Springer International Publishing, 2018.

[14] D. Jao and L. D. Feo. Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies. In B. Yang, editor, *Post-Quantum Cryptography - 4th International Workshop, PQCrypto 2011*, volume 7071 of *Lecture Notes in Computer Science*, pages 19–34. Springer, 2011.

[15] S. Jaques and J. M. Schanck. Quantum cryptanalysis in the RAM model: Claw-finding attacks on SIKE. In A. Boldyreva and D. Micciancio, editors, *Advances in Cryptology - CRYPTO 2019 - 39th Annual International Cryptology Conference Part I*, volume 11692 of *Lecture Notes in Computer Science*, pages 32–61. Springer, 2019.

[16] M. Knezevic, F. Vercauteren, and I. Verbauwhede. Speeding up bipartite modular multiplication. In M. A. Hasan and T. Helleseth, editors, *Arithmetic of Finite Fields, Third International Workshop, WAIFI 2010*, volume 6087 of *Lecture Notes in Computer Science*, pages 166–179. Springer, 2010.

[17] B. Koziel, R. Azarderakhsh, and M. Mozaffari-Kermani. Fast hardware architectures for supersingular isogeny Diffie-Hellman key exchange on FPGA. In O. Dunkelman and S. K. Sanadhya, editors, *Progress in Cryptology – INDOCRYPT 2016*, pages 191–206. Springer International Publishing, 2016.

[18] J. E. Ochoa-Jiménez. *Analysis and efficient implementation of public key cryptographic protocols*. PhD thesis, CINVESTAV-IPN, February 2019. Available at: http://delta.cs.cinvestav.mx/~francisco/je.pdf.

[19] N. I. of Standards and Technology. Submission requirements and evaluation criteria for the post-quantum cryptography standardization process, December 2016. https://csrc.nist.gov/Projects/Post-Quantum-Cryptography.

[20] C. Petit. Faster algorithms for isogeny problems using torsion point images. In T. Takagi and T. Peyrin, editors, *Advances in Cryptology - ASIACRYPT 2017 - 23rd International Conference on the Theory and Applications of Cryptology and Information Security Part II*, volume 10625 of *Lecture Notes in Computer Science*, pages 330–353. Springer, 2017.

[21] H. Seo, Z. Liu, P. Longa, and Z. Hu. SIDH on ARM: faster modular multiplications for faster post-quantum supersingular isogeny key exchange. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2018(3):1–20, 2018.