SIDH-sign: an efficient SIDH PoK-based signature

Jesús-Javier Chi-Domínguez[1], Víctor Mateu[1] and Lucas Pandolfo Perin[1]

Cryptography Research Centre, Technology Innovation Institute, Abu Dhabi, UAE
{jesus.dominguez,victor.mateu,lucas.perin}@tii.ae

Abstract. We analyze and implement the SIDH PoK-based construction from De Feo, Dobson, Galbraith, and Zobernigl. We improve the SIDH-PoK built-in functions to allow an efficient constant-time implementation. After that, we combine it with Fiat-Shamir transform to get an SIDH PoK-based signature scheme that we short label as SIDH-sign. We suggest SIDH-sign-p377, SIDH-sign-p546, and SIDH-sign-p697 as instances that provide security compared to NIST L1, L3, and L5. To the best of our knowledge, the three proposed instances provide the best performance among digital signature schemes based on isogenies.

Keywords: isogeny-based cryptography · signature scheme · proof-of-knowledge · implementation

1 Introduction

Recently, De Feo, Dobson, Galbraith, and Zobernig found an issue on the soundness proof for Supersingular Isogeny Diffie-Hellman (SIDH) [JD11, DJP14] based identification scheme [DDGZ21], which negatively impacts the signature schemes proposals from [YAJ+17] and [GPS20]. As a solution, [DDGZ21] presents an isogeny-based Proof of Knowledge (PoK) that relies on a new hardness assumption, and is immune to previously presented adaptive attacks [GPST16, BKM+20, DGL+20, FP22].

As a constructive implication of [DDGZ21], the authors hint at a non-interactive signature scheme using the Fiat-Shamir transformation, secure in the quantum random oracle model [LZ19]. It is worth mentioning that the resulting SIDH PoK-based signature works with large smooth isogenies (concerning powers of two and three), opposite to the proposals from [DG19, DPV19, EKP20, BKV19, DKL+20, BDK+21, DLW22] that require larger isogenies.

Related work. In 2019, De Feo and Galbraith proposed a signature scheme named SeaSign by combining the Commutative SIDH (CSIDH) [CLM+18] and Fiat-Shamir transformation with aborts [DG19]. SeaSign aims to have shorter keys than lattice signatures, but signing and verification are currently costly. Later, Decru, Panny, and Vercauteren improved SeaSign performance by allowing the prover not to answer a limited number of said parallel executions to decrease the rejection probability [DPV19]. Subsequently, Beullens, Kleinjung, and Vercauteren introduced a promising signature scheme labeled as CSI-FiSh [BKV19] by integrating similar optimizations of SeaSign on Stolbunov's signature scheme [Sto12]. They showed that including quadratic twists cuts the public key size in half, being 300 times faster and about three times smaller than any optimized version of SeaSign. In 2020, Kaafarani, Katsumata, and Pintore suggested a Lossy variant of CSI-FiSh with smaller signature sizes but two times slower than the original CSI-FiSh [EKP20].

A disadvantage of SeaSign, CSI-FiSh and its lossy variant, and the new scheme from [BDK+21], is that their current proposals and implementations use CSIDH-512, which

seems to bring lower quantum security than NIST Level 1 [BBP+21, Pei20, CSCJR21]. In particular, some state-of-the-art works [Pei20, CSCJR21] hint CSIDH instances with 2048 bits are good choices to close NIST Level 1 of security. Nevertheless, using large CSIDH instantiations (with about 2048 bits) would considerably slowdown on the performance and increase signature sizes for these CSIDH-based schemes, negatively impacting higher security levels compared to NIST Level 3 and 5.

Lastly, De Feo, Kohel, Leroux, Petit, and Wesolowski introduced a short-signature scheme called SQIsign [DKL+20]. They only target NIST level 1 of security, with signatures of 204 bytes, secret keys of 16 bytes, and public keys of 64 bytes; their C-code implementation claims 0.6 seconds for key generation, 2.5 seconds for signing, and 50 milliseconds for verification. Most recently, De Feo, Leroux, and Wesolowski showed that the security proof of SQISign is invalid; additionally, they fixed the issue and formulated a new computational assumption; they improved the signature computation in SQIsign by a factor of 2x [DLW22]. To the best of our knowledge, the current SQIsign signature implementation remains non-constant-time, and making it constant-time looks like a formidable task. It is still an open problem.

Contributions. For simplicity, we write SIDH-sign to a short naming of the SIDH PoK-based signature. We focus on describing and implementing the SIDH-sign signature, and list as follows our main contributions:

- we propose a uniform pseudo-random function to sample order-3^b kernel points;
- we present an efficient free-inverse and deterministic procedure of `CanonicalBasis` to compute SIDH public x-only order-3^b points for arbitrary Montgomery curves. Our proposal permits an efficient free-inverse entangled basis generation with shared Elligator (required in the compressed SIKE version) using precomputed tables of 32 bytes;
- we give an efficient SIDH-strategy based algorithm to find two-dimensional discrete logarithms on the 3^b-torsion subgroup for arbitrary Montgomery curves and without precomputated tables;
- we provide the first constant-time C-code implementation of the PoK from [DDGZ21]; and then
- we afford an efficient constant-time C-code implementation of SIDH-sign and suggest
  - SIDH-sign-p377 with desired NIST Level 1 (L1) of security,
  - SIDH-sign-p546 with desired NIST Level 3 (L2) of security, and
  - SIDH-sign-p697 with desired NIST Level 5 (L5) of security.

  Our security suggestions are obtained from the security analysis and SIKE proposals of [LWS21]. We also include benchmarks according to the NIST competition 3rd round SIKE parameter sets: SIDH-sign-p[434/503/610/751]. Our code is freely available at https://github.com/SIDH-sign/SIDH-Sign.

Outline. We organize the paper as follows. Section 2 gives an overview of the mathematical tools needed to understand the SIDH PoK-based construction. Section 3 presents our algorithmic proposals, which allow making the SIDH PoK practical, and Section 3.3 describes the SIDH-sign signature. Section 4 draws the benchmarks and comparisons with state-of-art isogeny-based signatures. Finally, Section 5 sums up our concluding remarks.

2 Preliminaries

In this section, we introduce all mathematical tools required in the SIDH PoK proposal from [DDGZ21]. Let $p = 2^a 3^b - 1$ be a prime number satisfying $p \equiv 3 \bmod 4$ for some $a, b \in \mathbb{Z}^+$. Let us denote a prime field with p elements as $\mathbb{F}_p$, and its quadratic field extension by $\mathbb{F}_{p^2} \coloneqq \mathbb{F}_p[i]/(i^2+1)$. Additionally, we set a supersingular curve E by means of Equation 1, and assume E has exactly $\#E(\mathbb{F}_{p^2}) = (p+1)^2$ points over $\mathbb{F}_{p^2}$.

$$E\colon y^2 = x^3 + Ax^2 + x, \quad A \in \mathbb{F}_{p^2}. \tag{1}$$

We denote the point at infinity of E as $\mathcal{O}$, which plays as the neutral element. We say $P \in E$ is an order-d point if d is the smallest positive integer such that

$$[d]P = \underbrace{P + \cdots + P}_{d \text{ times}} = \mathcal{O},$$

and write $E[d]$ to denote the d-torsion subgroup $\{P \in E(\mathbb{F}_{p^2}) \mid [d]P = \mathcal{O}\}$.

Isogenies From Kernel. An isogeny $\phi\colon E \to E'$ over $\mathbb{F}_{p^2}$ is a non-zero rational map fixing the point at infinity, $\phi(\mathcal{O}) = \mathcal{O}$. If such isogeny exists, we say E and E' are isogenous over $\mathbb{F}_{p^2}$, and this happens if and only if $\#E(\mathbb{F}_{p^2}) = \#E'(\mathbb{F}_{p^2})$. The kernel $\ker\phi$ of ϕ is the set $\{P \in E(\mathbb{F}_{p^2}) \mid \phi(P) = \mathcal{O}\}$. We only consider separable isogenies, and refer to ϕ as d-isogeny when $\#\ker\phi = d$ holds. Moreover, each d-isogeny is completely described by its kernel. The dual isogeny $\widehat{\phi}\colon E' \to E$ of ϕ satisfies that composing with ϕ gives the multiplication by d, that is

$$\widehat{\phi} \circ \phi\colon P \mapsto [d]P \quad \text{and} \quad \phi \circ \widehat{\phi}\colon P \mapsto [d]P.$$

Consequently, we have $\ker\left(\phi \circ \widehat{\phi}\right) = E[d]$ and $\ker\left(\widehat{\phi} \circ \phi\right) = E'[d]$. However, in practice, we only need to verify one composition. Given an order-d point $K \in E$, `IsogenyFromKernel`(K) denotes the procedure to compute the d-isogeny $\phi\colon E \to E'$ with $\ker\phi = \langle K \rangle$. Similarly, we use `DualKernel`$(\phi)$ to denote the computation of $\ker\widehat{\phi}$.

Kummer line. One beauty when working on Montgomery curves is that point arithmetic and isogenies can prevent y-coordinates and focus on x-only points (much cheaper operations [CH17, Ren18]). We use the notation $x(P)$ to refer to the x-coordinate of an elliptic curve point P. Formally, one works on the Kummer line $\{x(P) \mid P \in E\}$, and with x-only projective points $(X\colon Z)$ describing $x(P) = X/Z$ for some point $P \in E$.

2.1 SIDH-based PoK construction

Let `CanonicalBasis`(E) denotes the procedure to find two order-3^b points P and Q such that $\langle P, Q \rangle = E[3^b]$. SIDH-based PoK is a Proof-of-Knowledge that relies on isogenies over supersingular Montgomery curves, consisting of four algorithms: `commitment`, `challenge`, `response`, and `verification`. As setup, it has

- the quadratic field $\mathbb{F}_{p^2} = \mathbb{F}_p[i]/(i^2+1)$ along with $p = 2^a 3^b - 1$;
- the starting supersingular curve $E_0\colon y^2 = x^3 + 6x^2 + x$ [1]; and
- the order-3^b basis $\{P_0, Q_0\}$ satisfying $\langle P_0, Q_0 \rangle = E_0[3^b]$.

[1] We choose same E_0 as in SIKE proposal [ACC+20], but it can be a different curve

Given a public 2^a-isogenous curve E' to E. The prover wants to convince the verifier she knows the secret 2^a-isogeny $\phi\colon E \to E'$, which is equivalent to know $\ker\phi = \langle K_\phi \rangle$. Firstly, the prover creates a commitment as follows:

- Sample random 3^b-isogeny kernel $\langle K_\psi \rangle \subset E_0$;
- Write $K_\psi = [\alpha]P_0 + [\beta]Q_0 \in E_0$ for $\alpha, \beta \in \mathbb{Z}_{3^b}$;
- Compute $K_{\psi'} \leftarrow \phi(K_\psi)$ as $[\alpha]\phi(P_0) + [\beta]\phi(Q_0) \in E_1$;
- $\psi, E_2 \leftarrow \texttt{IsogenyFromKernel}(K_\psi)$;
- $P_2, Q_2 \leftarrow \texttt{CanonicalBasis}(E_2)$;
- $K_{\phi'} \leftarrow \psi(K_\phi) \in E_2$;
- $\phi', E_3 \leftarrow \texttt{IsogenyFromKernel}(K_{\phi'})$;
- $P_3, Q_3 \leftarrow \phi'(P_2), \phi'(P_2) \in E_3$;
- She then sends $\texttt{com} = (E_2, E_3, P_3, Q_3)$ to the verifier.

The verifier receives com, and sends a challenge $\texttt{chall} \xleftarrow{\$} \{0, 1\}$ to the prover. Subsequently, the prover responses

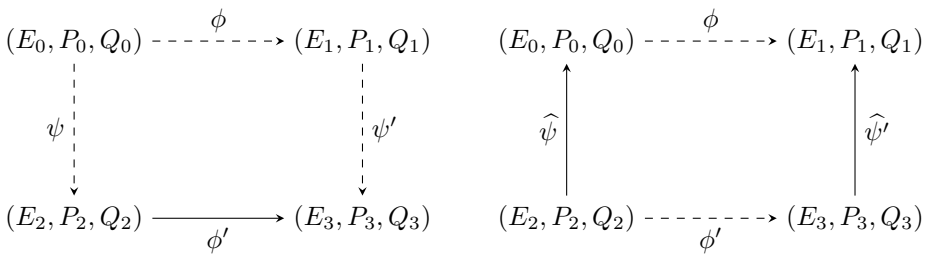
- **if** $\texttt{chall} = 1$ **then**
  - $\texttt{resp} \leftarrow K_{\phi'}$;
- **else**
  - $K_{\widehat{\psi}} \leftarrow \texttt{DualKernel}(\psi)$;
  - Write $K_{\widehat{\psi}} = [\gamma]P_2 + [\delta]Q_2 \in E_2$ for $\gamma, \delta \in \mathbb{Z}_{3^b}$;
  - $\texttt{resp} \leftarrow (\gamma, \delta)$;
- She sends resp to the verifier.

Lastly, the verifier receives resp and verifies as below:

- $(E_2, E_3, P_3, Q_3) \leftarrow \texttt{com}$
- **if** $\texttt{chall} = 1$ **then**
  - $K_{\phi'} \leftarrow \texttt{resp}$;
  - Check $K_{\phi'}$ has order 2^a and lies on E_2, otherwise output **reject**.
  - $P_2, Q_2 \leftarrow \texttt{CanonicalBasis}(E_2)$;
  - $\phi', E_3' \leftarrow \texttt{IsogenyFromKernel}(K_{\phi'})$;
  - Verify $E_3 = E_3'$ and $P_3 = \phi'(P_2)$, $Q_3 = \phi'(Q_2)$ otherwise output **reject**.
- **else**
  - $(\gamma, \delta) \leftarrow \texttt{resp}$;
  - $P_2, Q_2 \leftarrow \texttt{CanonicalBasis}(E_2)$;
  - $K_{\widehat{\psi}} \leftarrow [\gamma]P_2 + [\delta]Q_2 \in E_2$;
  - $K_{\widehat{\psi'}} \leftarrow [\gamma]P_3 + [\delta]Q_3 \in E_3$;

  - Check $K_{\widehat{\psi}}, K_{\widehat{\psi}'}$ have order 3^b, otherwise output **reject**.
  - $\widehat{\psi}, E_0' \leftarrow \texttt{IsogenyFromKernel}(K_{\widehat{\phi}})$;
  - $\widehat{\psi}', E_1' \leftarrow \texttt{IsogenyFromKernel}(K_{\widehat{\phi}})$;
  - Verify $E_0 = E_0'$ and $E_1 = E_1'$, otherwise output **reject**.
- Output **accept**.

As a way to describe the security assumption, Figure 1 illustrates the hard problem of the above PoK, and assumes the cases from Figure 1a and Figure 1a do not simultaneously occur for a fixed instance. Essentially, the hardness assumption relies on distinguishing between well-formed and altered instances (E_2, E_3, P_3, Q_3) [DDGZ21, §2.1 and §3.2], and on the computational hardness of finding $\phi\colon E_0 \to E_1$.



(a) Given $\ker \phi' = \langle K_\phi' \rangle$. The verifier accepts whenever ϕ' is a 2^a-isogeny with co-domain curve equal E_3, $\phi'(P_2) = P_3$, and $\phi'(Q_2) = Q_3$.

(b) Given $\ker \widehat{\psi} = \langle [\gamma]P_2 + [\delta]Q_2 \rangle$ and $\ker \widehat{\psi}' = \langle [\gamma]P_3 + [\delta]Q_3 \rangle$. The verifier accepts whenever $\widehat{\psi}$ and $\widehat{\psi}'$ are 3^b-isogenies with co-domain curves equal E_0 and E_1, respectively.

Figure 1: The points $P_2, Q_2 \leftarrow \texttt{CanonicalBasis}(E_2)$ are computed on the fly. Dashed arrows are secret and unknown by the adversary and distinguisher.

2.2 Sigma protocol & the Fiat-Shamir transform

The sigma protocol described by Section 2.1 is 2-special sound under the relation

$$\mathcal{R}_{\texttt{weakSIDH}} \coloneqq \{(E_1, \psi) \mid \psi\colon E_0 \to E_1 \text{ is a } 2^a\text{-isogeny}\}\,.$$

When repeated κ times, it becomes a Honest-Verifier Zero-Knowledge (HVZK) PoK with soundness $2^{-\kappa}$ [DDGZ21].

Notice that this PoK proves knowledge of the private key related to a public key as an interactive identification protocol. These protocols can be converted into digital signature schemes using the strong Fiat-Shamir transform [FS86, BPW12]. The main idea is to avoid the interaction between the prover and the verifier by allowing the prover to generate the challenge as the hash of the statement and the commitment. In our case, the prover would first generate κ commitments $\texttt{com}_i$ and then obtains the challenge $\texttt{chall} = \mathcal{H}(\texttt{pk}, \texttt{com}_0, \ldots, \texttt{com}_{\kappa-1})$. We denote with $\texttt{pk}$ an SIDH public key composed of the triplet: (E_1, P_1, Q_1). Each bit of $\texttt{chall}$ represents the challenge $\texttt{chall}_i$ that determines the response values for $\texttt{com}_i$. This transformation has been proven secure [Unr17] in the Quantum Random Oracle Model (QROM).

3 Efficient SIDH-PoK built-in function proposals

This section centers on describing the way we efficiently did each computation from Section 2.1. Let us start with the sampling of a random 3^b-isogeny kernel K_ψ. Let $\langle P, Q\rangle = E[3^b]$, then the number of different order-3^b subgroups $\langle R\rangle \subset E[3^b]$ is $4 \times 3^{b-1}$. In fact, either $\langle R\rangle$ coincides with $\langle P + [k]Q\rangle$ or $\langle [k]P + Q\rangle$ for some integer $k \in [\![0 \,..\, 3^b - 1]\!]$. Furthermore, Equation 2 give us a bijection between $\{0, 1, 2, 3\} \times [\![0 \,..\, 3^{b-1} - 1]\!]$ and the set $\mathcal{A}$ of order-3^b subgroups of $E[3^b]$.

$$f\colon (o, k) \mapsto \begin{cases} \langle P + [o3^{b-1} + k]Q\rangle & \text{if } o \neq 3, \\ \langle [3k]P + Q\rangle & \text{otherwise.} \end{cases} . \tag{2}$$

There is a uniform pseudo-random function to sample elements on the set $\mathcal{A}$ of order-3^b subgroups of $E[3^b]$. The idea is to construct a uniform pseudo-random function $g\colon \mathbb{Z}^+ \to \{0, 1, 2, 3\} \times [\![0 \,..\, 3^{b-1} - 1]\!]$, then compose it with $f\colon \{0, 1, 2, 3\} \times [\![0 \,..\, 3^{b-1} - 1]\!] \to \mathcal{A}$ and get $h = f \circ g$ as desired. In particular, g, as described by Algorithm 1, makes the magic for us.

Algorithm 1 Pseudo-random function $g\colon \mathbb{Z}^+ \to \{0, 1, 2, 3\} \times [\![0 \,..\, 3^{b-1} - 1]\!]$

Inputs: A positive integer b
Output: A random element $(o, k) \in \{0, 1, 2, 3\} \times [\![0 \,..\, 3^{b-1} - 1]\!]$
1: `B` $\leftarrow$ number of bytes of 3^{b-1}
2: **repeat**
3: `var` $\leftarrow$ (`B` + 1) random bytes using `SHAKE256` as a PRG
4: $k \leftarrow$ `B` least significant bytes of `var`
5: $o' \leftarrow$ most significant byte of `var`
6: **until** $k < 3^{b-1}$
7: $o \leftarrow$ two least significant bits of o'
8: **return** (o, k)

Dual Kernel. Let us sample (o, k) and $\langle R\rangle$ by using Algorithm 1 and Equation 2. Let ψ be a 3^b-isogeny with kernel $\langle R\rangle$,

$$\widehat{f}\colon o \mapsto \begin{cases} \langle \psi(Q)\rangle & \text{if } o \neq 3, \\ \langle \psi(P)\rangle & \text{otherwise.} \end{cases} .$$

and $\widehat{\psi}$ be the 3^b-isogeny with kernel $\widehat{R} = \widehat{f}(o)$. By construction, $\psi \circ \widehat{\psi}$ has kernel equal to $\langle P, Q\rangle = E[3^b]$, and thus $\widehat{f}$ describes the dual 3^b-isogeny kernel concerning ψ.

3.1 Canonical Basis

Notice Equation 1 has a projective representation given by Equation 3, where $x = X/Z \in \mathbb{F}_{p^2}$ and $y = Y/Z \in \mathbb{F}_{p^2}$ satisfies Equation 1.

$$E\colon Y^2Z^2 = X^3Z + AX^2Z^2 + XZ^3 \tag{3}$$

Thus, given X and Z we can check the existence of $Y \in \mathbb{F}_{p^2}$ by determining whether the right-hand side of Equation 3 is QR[2][3]. Now, following the ideas from [CCC+19], we next give an efficient description of Elligator (without inverse computations): let's

[2] We write QR to refer quadratic residue. Similarly, QNR means quadratic non-residue.
[3] The QR check of an element $(t_0 + t_1 i) \in \mathbb{F}_{p^2}$ is efficiently done via the QR test of its norm $t_0^2 + t_1^2$ over $\mathbb{F}_p$ [CJL+17].

assume $A \neq 0$, and let $u \in \mathbb{F}_{p^2}$ be a QR that does not belong $\mathbb{F}_p$ and $r \in \mathbb{F}_{p^2}$, then, if $(X : Z) = (-A : 1 + ur^2)$ ensure existence of Y and X/Z is a QNR, there is also a Y' for $(X' : Z') = (-Aur^2 : 1 + ur^2)$ [PDJ21]. Algorithm 2 applies the above ideas to deterministically get a projective x-only point $(X : Z)$ on E by using the below equation

$$X^3Z + AX^2Z^2 + XZ^3 = A \cdot (1 + ur^2) \cdot \left[A^2 \cdot (ur^2) - (1 + ur^2)^2\right].$$

Take note that X/Z is QRN if and only if XZ too; furthermore, we require Z such that i) A is QNR and Z is QR, or ii) A is QR and Z is QNR.

Algorithm 2 `Elligator2` procedure without field inversion

Inputs: A-coefficient of a supersingular curve E given by Equation 1, an element $r \in \mathbb{F}_p$, and a QR $u \in \mathbb{F}_{p^2}$ that does not belong $\mathbb{F}_p$
Output: A projective x-only point $(X : Z)$ on E, and a point-shape identifier qr $\in \{0, 1\}$

1: $Z \leftarrow ur^2$
2: $X' \leftarrow AZ$
3: $Y' \leftarrow AX'$
4: $s \leftarrow A + X'$
5: $Z \leftarrow Z + 1$
6: $t \leftarrow Z^2$
7: $Y' \leftarrow Y' - t$
8: $Y' \leftarrow sY'$
9: **if** Y' is QR **then**
10: $\quad X \leftarrow -A$
11: $\quad$ qr $\leftarrow 1$
12: **else**
13: $\quad X \leftarrow -X'$
14: $\quad$ qr $\leftarrow 0$
15: **end if**
16: **return** $(X : Z)$, qr

Recall, we do not have any restriction for r, and for efficiency, we can choose small values of $r \in \mathbb{F}_p$. In fact, for a fixed u we suggest to precompute two lists `QR` and `QNR` of 32 bytes that describe 8-bits integers $r \in \mathbb{F}_p$ satisfying $Z = 1 + ur^2$ is a QR or not, respectively. It is worth to mention, $(X : Z) = (-A : 1 + ur^2)$ and $(X' : Z') = (-Aur^2 : 1 + ur^2)$ share the same Z-coordinate, and permit to efficiently compute $x(P - Q)$ by using Equation 4.

$$x(P - Q) = \frac{[x(P)^3 + Ax(P)^2 + x(P)](1 + r\sqrt{u})^2}{[x(P) - x(Q)]^2}, \tag{4}$$

where $x(P) = X/Z$ and $x(Q) = X'/Z$. Algorithm 3 summarizes how to get $x(P - Q)$ with projective x-only points as inputs. Finally, Algorithm 4 describes an efficient implementation of `CanonicalBasis`. Lemma 1 illustrates why Algorithm 4 works.

Lemma 1. *Let $\mathbb{F}_{p^2} = \mathbb{F}_p[i]/(i^2 + 1)$ where $p = 2^a3^b - 1$ is a prime number, A-coefficient of a supersingular curve E given by Equation 1, $u = (i + 1)^2 \in \mathbb{F}_{p^2}$, and let* `QR` *and* `QNR` *two lists of 32 bytes that describe 8-bits elements $r \in \mathbb{F}_p$ satisfying $Z = 1 + ur^2$ is a QR or not, respectively. Then, Algorithm 4 correctly gets the projective x-only points $x(P)$, $x(Q)$, and $x(P - Q)$ such that $E[3^b] = \langle P, Q\rangle$.*

Proof. Let $r \in \mathbb{F}_{p^2}$ be as in Line 8 from Algorithm 4, and let $x(P)$, $x(Q)$, and $x(P - Q)$

Algorithm 3 `get_P_minus_Q` procedure without field inversion

Inputs: A-coefficient of a supersingular curve E given by Equation 1, $x(P) \coloneqq (X : Z) = (-A : 1+ur^2)$, $x(Q) \coloneqq (X' : Z') = (-Aur^2 : 1+ur^2)$, the element $r \in \mathbb{F}_p$, and $u' = \sqrt{u}$
Output: The projective x-only point $x(P-Q) = (\overline{X} : \overline{Z})$

1: $t \leftarrow u'r$
2: $t \leftarrow t+1$
3: $t \leftarrow t^2$
4: $\overline{X} \leftarrow X^2$
5: $\overline{Z} \leftarrow Z^2$
6: $s \leftarrow A\overline{X}$
7: $s \leftarrow sZ$
8: $x \leftarrow X\overline{Z}$
9: $x \leftarrow s+x$
10: $\overline{X} \leftarrow \overline{X}X$
11: $x \leftarrow \overline{X}+x$
12: $\overline{X} \leftarrow \overline{X}t$
13: $t \leftarrow X - X'$
14: $t \leftarrow t^2$
15: $\overline{Z} \leftarrow tZ$
16: **return** $(\overline{X} : \overline{Z})$

be the output of Algorithm 4. Then,

$$\Pr\left[[2^a]P \text{ has order } 3^b\right] = \Pr\left[[2^a3^{b-1}]P \neq \mathcal{O}\right] = \frac{3}{4}, \quad \text{and}$$
$$\Pr\left[[2^a]Q \text{ has order } 3^b\right] = \Pr\left[[2^a3^{b-1}]Q \neq \mathcal{O}\right] = \frac{3}{4}.$$

Now, assuming P and Q behave as independent random points, we have

$$\Pr[P \text{ and } Q \text{ have order } 3^b] = \Pr\left[[2^a3^{b-1}]P \neq \mathcal{O}\right] \times \Pr\left[[2^a3^{b-1}]Q \neq \mathcal{O}\right] = \frac{9}{16}.$$

Therefore, the probability of success in one iteration of Algorithm 4 becomes

$$\begin{aligned}\Pr\left[r \text{ gives an order-}3^b \text{ basis } \{P,Q\}\right] &= \Pr[\text{Line 9 returns } x(P) = (-A : 1+ur^2)] \\ &\quad \times \Pr[P \text{ and } Q \text{ have order } 3^b] \\ &\quad \times \Pr\left[[2^a3^{b-1}]P \neq \pm[2^a3^{b-1}]Q\right] \\ &= \left(\frac{1}{2}\right)\left(\frac{9}{16}\right)\left(\frac{3}{4}\right) = \frac{27}{128}.\end{aligned}$$

That is, we expect (in average) $\frac{32\times 27}{128} = 6.75$ success from 32 random samples, which implies Algorithm 4 correctly gets an order-3^b point basis. □

3.2 Decomposition by scalars

We follow the strategy approach of computing ℓ^e isogenies, which was recently applied to solving two dimensional discrete logarithms [PB21]. The main difference in our case study is that we must compute two dimensional discrete logarithms on a random curve E; that is, we want to find $\gamma, \delta \in [\![0 \,..\, 3^b - 1]\!]$ given $R = [\gamma]P + [\delta]Q$. Our approach does not use precomputed tables, it only requires a strategy coded as in SIDH:

- a list S_b of $b-1$ positive integers, and

Algorithm 4 `CanonicalBasis` procedure

Inputs: prime number $p = 2^a 3^b - 1$, A-coefficient of a supersingular curve E given by Equation 1, $u = (i+1)^2 \in \mathbb{F}_{p^2}$, and two lists `QR` and `QNR` of 32 bytes that describe 8-bits elements $r \in \mathbb{F}_p$ satisfying $Z = 1 + ur^2$ is a QR or not, respectively.
Output: The projective x-only points $x(P)$, $x(Q)$, and $x(P-Q)$ on E s.t. $E[3^b] = \langle P, Q \rangle$

1: **if** A is QR **then**
2: $T \leftarrow$ `QNR`
3: **else**
4: $T \leftarrow$ `QR`
5: **end if**
6: $u' \leftarrow i + 1$
7: **for** t in T **do**
8: $r \leftarrow i + t$
9: $x(P), \texttt{shape} \leftarrow \texttt{Eligator2}(A, r, u)$ // $(X : Z) = x(P)$
10: **if** `shape` $= 0$ **then**
11: **skip** this iteration, and **go to** next iteration
12: **end if**
13: $x(Q) \leftarrow (-Aur^2 : Z)$
14: **if** `shape` $= 0$ **then**
15: **skip** this iteration, and **go to** next iteration
16: **end if**
17: $x(P-Q) \leftarrow \texttt{get\_P\_minus\_Q}(A, x(P), x(Q), r, u')$
18: **if** $[2^a]P$ or $[2^a]Q$ has order smaller than 3^b **then**
19: **skip** this iteration, and **go to** next iteration
20: **end if**
21: **if** $[2^a 3^{b-1}]P = \pm[2^a 3^{b-1}]Q$ **then**
22: **skip** this iteration, and **go to** next iteration
23: **end if**
24: **finalize** loop iterations, and **go to** Line 26 // At this point, $\langle P, Q \rangle = E[3^b]$
25: **end for**
26: **return** $x(P)$, $x(Q)$, $x(P-Q)$

- each entry $S_b[j]$ determines the number of point triplings before including a new point on the computations.

As pointed out in [PB21], we require to perform point additions (not only-x points) as efficiently as possible. So, we also translate the problem in E into an isomorphic Twisted Edwards curve and use projective coordinates $(X : Y : Z)$ to ensure efficient point additions, doubling, and triplings. Let us write $\gamma = \sum_{j=0}^{b-1} \gamma^j$, and $\delta = \sum_{j=0}^{b-1} \delta_j 3^j$. For each $b' \in [\![0 \,..\, b-1]\!]$ we have

$$\left[3^{b-1-b'}\right]\left(R - \sum_{j=0}^{b'-1}[\gamma_j 3^j]P - \sum_{j=0}^{b'}[\delta_j 3^j]Q\right) = [\gamma_{b'}]\left(\left[3^{b-1}\right]P\right) + [\delta_{b'}]\left(\left[3^{b-1}\right]Q\right)$$
$$= \begin{cases} \mathcal{O} \\ \pm\left[3^{b-1}\right]P \\ \pm\left[3^{b-1}\right]Q \\ \pm\left[3^{b-1}\right](P+Q) \\ \pm\left[3^{b-1}\right](P+[2]Q). \end{cases} \tag{5}$$

Next, strategy S_b describes a weighted subgraph contained into a discrete rectangular

triangle Δ of size b, and then it has a cost equal to

$$\texttt{cost}(S_b) = \texttt{cost}(S_h) + \texttt{cost}(S_{b-h}) + (n-h)\cdot q + h \cdot p,$$

where $1 \leq h \leq b-1$, and p determines a point tripling cost while q the cost for computing $c_{b'}$ and $d_{b'}$ both in $\{0,1,2\}$. Notice, we have 9 options for $(c_{b'}, d_{b'}) \in \{(0,0),(1,1),(2,2),(0,1),(0,2),(1,2),(1,0),(2,0),(2,1)\}$. So, assuming uniform distribution on the choices of $(c_{b'}, d_{b'})$, we suggest to set q as the average cost of these nine possibilities. We say S_b is optimal if it has the minimum possible cost. Algorithm 5 draws our strategy approach and tries to iteratively find each coefficient $\gamma_{b'}$ and $\delta_{b'}$ by reducing into Equation 5 and solving it (as opposite to 3^b-isogenies case, which looks for the 3-isogeny kernel). In summary, the strategy evaluation concerning S_b has a running time of $O(b\log_2(b))$ point operations and essentially shares the same structure (procedure) as computing 3^b-isogenies.

Algorithm 5 two dimensional discrete logarithm computation based on SIDH-like strategies

Inputs: two linearly independent order-3^b points P and Q, a point $R \in \langle P, Q\rangle$, and an strategy S_b coded as a list of $b-1$ positive.
Output: $\gamma = \sum_{j=0}^{b-1}\gamma_j 3^j$ and $\delta = \sum_{j=0}^{b-1}\delta_j 3^j \in \mathbb{Z}_{3^b}$ such that $R = [\gamma]P + [\delta]Q$

$\gamma \leftarrow 0$
$\delta \leftarrow 0$
$\texttt{points} \leftarrow []$
$\texttt{indexes} \leftarrow []$
$\texttt{index} \leftarrow 0$
$k \leftarrow 0$
for $\texttt{row} \coloneqq 1$ to $b-1$ **do**
  while $\texttt{index} < b - \texttt{row}$ **do**
    $\texttt{indexes}[n] \leftarrow \texttt{index}$
    $\texttt{points}[n] \leftarrow R$
    $R \leftarrow \left[3^{S_b[k]}\right] R$
    $\texttt{index} \leftarrow \texttt{index} + S_b[k]$
    $k \leftarrow k + 1$
    $n \leftarrow n + 1$
  end while // This while loop computes the left-hand side of Equation 5
  Get $\gamma_{\texttt{row}-1}$ and $\delta_{\texttt{row}-1}$ by means of Equation 5
  for $j \coloneqq 0$ to $n-1$ **do**
    $\texttt{points}[j] \leftarrow \texttt{points}[j] - \left[3^{\texttt{row}-1+\texttt{indexes}[j]}\right]([\gamma_{\texttt{row}-1}]P + [\delta_{\texttt{row}-1}]Q)$
  end for
  $R \leftarrow \texttt{points}[n-1]$
  $\texttt{index} = \texttt{indexes}[n-1]$
  $n \leftarrow n - 1$
end for
Get γ_{b-1} and δ_{b-1} by means of Equation 5
$\gamma \leftarrow \sum_{j=0}^{b-1}\gamma_j 3^j$
$\delta \leftarrow \sum_{j=0}^{b-1}\delta_j 3^j$
return γ, δ

Two dimensional ladder. Let's assume we have the following task: given $\gamma, \delta \in \mathbb{Z}_{3^b}$ and $P, Q \in E$, we want to compute $[\gamma]P + [\delta]Q$. Notice that verification is the only place where double scalar multiplication is needed, so one of the best ways to compute it is through the Euclidean two dimensional algorithm [CS18, Algorithm 9].

Recovery of y-coordinate. Since all the computations on the Montgomery curves are concern x-only points, once we need to move to Twisted Edwards curves, we use Okeya-Sakurai y-coordinate recovery formula as in [CS18, §4.3].

3.3 Explicit description of SIDH-sign

As mentioned in Section 2.2, a digital signature scheme comes from the PoK of De Feo et al. [DDGZ21]. Their original contribution gives insights on how to do this by using the Fiat-Shamir transform and provide numbers for the sizes of the resulting scheme. For precision, we next provide a clear idea of the different algorithms involved in the digital signature.

A digital signature scheme consist of three main algorithms: $\texttt{keygen}(\lambda)$, $\mathsf{Sign}(\mathsf{pk}, \mathsf{sk}, m)$, and $\mathsf{Verify}(\mathsf{pk}, \sigma, m)$. The first algorithm receives as input a security parameter λ in order to generate a key pair with the appropriate parameters to later generate digital signatures with security at least 2^λ. The operations involved are detailed in Figure 2 and easily extracted from the setup of the SIDH-PoK.

The second algorithm $\mathsf{Sign}(\mathsf{pk}, \mathsf{sk}, m)$ generates a digital signature σ of the message m using the secret key sk and the public key pk required for the PoK. In Figure 2 we detail the specifics of the algorithm. Basically, it can be seen as a repetitive call to the algorithm `commitment`, from the PoK, then a challenge is computed by applying the Fiat-Shamir transformation with the generated commitments and, finally, an iterative call to `response` using the bits of the challenge is required to compute the last values of the signature.

In a similar way, algorithm $\mathsf{Verify}(\mathsf{pk}, \sigma, m)$ generates the challenge using the commitments in σ, the public key pk, and the message m. After that, it uses the bits of the generated challenge and the responses in σ as inputs for the algorithm `verification` from the PoK. If all the iterations are correct, the algorithm $\mathsf{Verify}(\mathsf{pk}, \sigma, m)$ outputs accept. More details of this algorithm are provided in Figure 3.

4 Experiments

In our implementation, all computations are with x-only projective points (omitting the decomposition by scalars of Section 3.2). We take as the base 2^a-isogenies and 3^b-isogenies implementation from the SIDH implementation. We apply the improvements of [EKA22] when a is odd. Consequently, we work with commitments as $(E_2, x(P_3), x(Q_3), x(P_3 - Q_3))$ instead of (E_2, E_3, P_3, Q_3). In both signing and verifying, we use `SHAKE256` as the hash function $\mathcal{H}$.

Now, SIDH-sign has private and public keys, and signatures of $\left(\frac{\log_2(p)}{2}\right)$-bits and $6\log_2(p)$ (as in SIKE), and at most $10\log_2(p)\kappa$-bits where $\kappa \in \{128, 160, 192, 256\}$, respectively; Table 1 encloses these sizes for each SIDH-sign instance. Notice, the case $\texttt{chall} = 1$ saves $\frac{1}{10}$ of the bytes compared with $\texttt{chall} = 0$, and thus the minimum signature size becomes of $\frac{19}{20} \times 10\log_2(p)\kappa = 9.5\log_2(p)\kappa$-bits. Furthermore, in average we expect a signature size of $9.75\log_2(p)\kappa$ bits. Table 1 lists the sizes (in bytes) of different SIDH-sign instantiations.

We replicate the benchmark experiments of SQISign in [DLW22] by using its publicly available C-code implementation, to provide a baseline for comparing the performance of SIDH-sign. Furthermore, we recall that while the SQISign implementation is not constant-time, our C-code implementation of SIDH-sign is based on the constant-time SIKE implementation. Indeed, we provide a constant-time implementation of all our results from Section 3, and apply Longa's tricks from [Lon22] to speed up the field field arithmetic. We refactor the SIKE code to avoid unused variables in the PoK construction and make it as small as possible. We also split the Alice and Bob computations into different curve structures (for easy debugging). We do not really improve internal computations, we (in

$\underline{\texttt{keygen}(\lambda)}$

Sample random 2^a-isogeny kernel $\langle K_\phi \rangle \subset E_0$

$\phi, E_1 \leftarrow \texttt{IsogenyFromKernel}(K_\phi)$

$P_1, Q_1 \leftarrow \phi(P_0), \phi(Q_0) \in E_1$

$\mathsf{pk} = (E_1, P_1, Q_1)$

$\mathsf{sk} = (\phi, K_\phi)$

return $(\mathsf{pk}, \mathsf{sk})$

$\underline{\mathsf{Sign}(\mathsf{pk}, \mathsf{sk}, m)}$

for $i = 1$ to κ **do**

  Sample random 3^b-isogeny kernel $\langle K_{\psi_i} \rangle \subset E_0$

  Find α_i, β_i s. t. $K_{\psi_i} = [\alpha_i]P_0 + [\beta_i]Q_0 \in E_0$

  $K_{\psi_i'} = \phi(K_{\psi_i})$

  $\psi_i, E_{2,i} \leftarrow \texttt{IsogenyFromKernel}(K_{\psi_i})$

  $P_{2,i}, Q_{2,i} \leftarrow \texttt{CanonicalBasis}(E_{2,i})$

  $K_{\phi_i'} \leftarrow \psi_i(K_\phi)$

  $\phi_i', E_{3,i} \leftarrow \texttt{IsogenyFromKernel}(K_{\phi_i'})$

  $P_{3,i}, Q_{3,i} \leftarrow \phi_i'(P_{2,i}), \phi_i'(P_{2,i}) \in E_{3,i}$

  $\texttt{com}_i = (E_{2,i}, E_{3,i}, P_{3,i}, Q_{3,i})$

end

$\texttt{chall} = \mathcal{H}(\mathsf{pk}, \texttt{com}_1, \ldots, \texttt{com}_\kappa)$

for $i = 1$ to κ **do**

  if $\texttt{chall}[i] = 1$ **do**

    $\texttt{resp}_i = K_{\phi_i'}$

  else

    $K_{\widehat{\psi_i}} \leftarrow \texttt{DualKernel}(\psi_i)$

    Find γ_i, δ_i s. t. $K_{\widehat{\psi_i}} = [\gamma_i]P_{2,i} + [\delta_i]Q_{2,i} \in E_{2,i}$

    $\texttt{resp}_i = (\gamma_i, \delta_i)$

  end

end

return $\sigma = (\texttt{com}_1, \ldots, \texttt{com}_\kappa, \texttt{resp}_1, \ldots, \texttt{resp}_\kappa)$

Figure 2: Key Generation and signature algorithms. The public parameters are the curve E_0, and the points $P_0, Q_0 \in E_0$ satisfying $\langle P_0, Q_0 \rangle = E_0[3^b]$.

some sense) profile and clean their code for our case study. For compatibility, our prime field arithmetic is based on CSIDH and SQIsign codes and allows us to integrate different primes easily.

The running time of SIDH-sign depends on the public challenge and is independent from the private key. Table 2 draws the L1/L2 SQIsign-p3923 and SIDH-sign-p[377/434/503] timings when generating key pairs, signing, and verifying. While Table 3 focuses on large instantiations of SIDH-sign aimed at L3 and L5 parameters. All of our experiments were executed on Ubuntu 21.04 running on an Intel Core i9-10885h machine with 32GB of RAM

Verify(pk, σ, m)

Parse σ as $\sigma = (\mathtt{com}_1, \ldots, \mathtt{com}_\kappa, \mathtt{resp}_1, \ldots, \mathtt{resp}_\kappa)$
$\mathtt{chall} = \mathcal{H}(\mathsf{pk}, \mathtt{com}_1, \ldots, \mathtt{com}_\kappa)$
for $i \in 1, \ldots, \kappa$ **do**
  $(E_{2,i}, E_{3,i}, P_{3,i}, Q_{3,i}) \leftarrow \mathtt{com}_i$
  if $\mathtt{chall}[i] = 1$ **do**
    $K_{\phi'_i} \leftarrow \mathtt{resp}_i$
    if $K_{\phi'_i} \notin E_{2,i}$ and $K_{\phi'_i}$ order $\neq 2^a$ **do**
      return reject
    end
    $P_{2,i}, Q_{2,i} \leftarrow \mathtt{CanonicalBasis}(E_{2,i})$
    $\phi'_i, E'_{3,i} \leftarrow \mathtt{IsogenyFromKernel}(K_{\phi'_i})$
    if $E_{3,i} \neq E'_{3,i}$ or $P_{3,i} \neq \phi'_i(P_{2,i})$, or $Q_{3,i} \neq \phi'_i(Q_{2,i})$ **do**
      return reject
    end
  else
    $(\gamma_i, \delta_i) \leftarrow \mathtt{resp}_i$
    $P_{2,i}, Q_{2,i} \leftarrow \mathtt{CanonicalBasis}(E_{2,i})$
    $K_{\widehat{\psi}_i} \leftarrow [\gamma_i]P_{2,i} + [\delta_i]Q_{2,i} \in E_{2,i}$
    $K_{\widehat{\psi'_i}} \leftarrow [\gamma_i]P_{3,i} + [\delta_i]Q_{3,i} \in E_{3,i}$
    if $K_{\widehat{\psi}_i}$ or $K_{\widehat{\psi'_i}}$ does not have order 3^b **do**
      return reject
    end
    $\widehat{\psi}_i, E'_{0,i} \leftarrow \mathtt{IsogenyFromKernel}(K_{\widehat{\psi}_i})$
    $\widehat{\psi'_i}, E'_{1,i} \leftarrow \mathtt{IsogenyFromKernel}(K_{\widehat{\psi'_i}})$
    if $E_0 \neq E'_{0,i}$ or $E_1 \neq E'_{1,i}$ **do**
      return reject
    end
  end
end
return accept

Figure 3: Verification algorithm. The public parameters are the curve E_0, and the points $P_0, Q_0 \in E_0$ satisfying $\langle P_0, Q_0 \rangle = E_0[3^b]$.

Table 1: Byte sizes. Signature sizes correspond with the maximum (when $\mathtt{chall} = 0$).

	Security Level	Private key	Public Key	Signature
SIDH-sign-p377	NIST L1 ($\kappa = 128$)	24 B	288 B	61.44 KB
SIDH-sign-p546	NIST L3 ($\kappa = 192$)	34 B	414 B	132.48 KB
SIDH-sign-p697	NIST L5 ($\kappa = 256$)	44 B	528 B	225.28 KB
SIDH-sign-p434	NIST L1 ($\kappa = 128$)	27 B	330 B	70.4 KB
SIDH-sign-p503	NIST L2 ($\kappa = 160$)	31 B	378 B	100.80 KB
SIDH-sign-p610	NIST L3 ($\kappa = 192$)	38 B	462 B	147.84 KB
SIDH-sign-p751	NIST L5 ($\kappa = 256$)	47 B	564 B	240.64 KB

and clang 12.0.0 compiler. During the experiment the turbo boost and hyper-threading settings were disabled and CPU clock was fixed at 2.0GH. Our code is freely available at https://github.com/SIDH-sign/SIDH-Sign.

Table 2: Timings measured in *milliseconds* and *millions of cycles* for SQIsign-p3923 (L1) compared to SIDH-sign-p377 (L1), SIDH-sign-p434 (L1) and SIDH-sign-p503 (L2). Q1 refers to the 1st quartile, and Q3 is the 3rd quartile.

		SQIsign-p3923			SIDH-sign-p377			SIDH-sign-p434			SIDH-sign-p503		
		keygen	Sign	Verify	keygen	Sign	Verify	keygen	Sign	Verify	keygen	Sign	Verify
Mcycles	Q1	789	3,987	77	5	3,133	1,816	7	4,405	2,543	10	7,859	4,455
	median	819	4,028	78	5	3,213	1,859	7	4,521	2,594	10	8,002	4,528
	Q3	851	4,099	82	5	3,265	1,892	7	4,603	2,642	10	8,122	4,598
ms	Q1	329	1,661	32	2	1,305	756	3	1,835	1,059	4	3,274	1,856
	median	341	1,678	32	2	1,339	774	3	1,883	1,081	4	3,334	1,886
	Q3	354	1,708	34	2	1,360	788	3	1,918	1,101	4	3,384	1,916

In the following, we discuss the result of the baseline experiment shown in Table 2 and Table 3, observing the median of cycles for each operation. The key generation for SIDH-sign-p377 is 149 times faster than SQISign-p3923, at a trade-off where signature verification is 23.7 times slower. Remarkably, we also achieve signature generation 1.25 times faster than SQISign using slower and constant-time code. Considering larger primes such as p434 and p502, Table 2 shows that only key generation preserves the performance advantage over SQISign, with greater trade-offs to signature verification cost and in addition to slower signature generation. Moreover, at the same level of security, p377 offers much better performance when compared to p434, running ~ 1.4 times faster for all operations. Indeed, this performance gain is identical when comparing NIST L3 parameters p546 and p610 in Table 3. For NIST L5 parameters, p697 is 2.05 times faster for key generation and ~ 2.15 times faster for signing and verifying, when compared to p751. The main reason for the slower performance of p751 is that it is the only parameter set where the optimizations to the prime field arithmetic in [Lon22] where not included. This is due to an incompatibility regarding our code API and the length of the prime number.

Table 3: Timings measured in *milliseconds* and *millions of cycles* concerning larger parameter sets (L3 ad L5). Q1 refers to the 1st quartile, and Q3 is the 3rd quartile.

		SIDH-sign-p546			SIDH-sign-p610			SIDH-sign-p697			SIDH-sign-p751		
		keygen	Sign	Verify	keygen	Sign	Verify	keygen	Sign	Verify	keygen	Sign	Verify
Mcycles	Q1	13	12,181	6,982	19	17,440	9,822	27	30,660	17,740	56	66,353	38,141
	median	13	12,334	7,075	19	17,649	9,977	27	31,188	18,011	56	67,400	38,606
	Q3	13	12,541	7,161	19	17,942	10,101	27	31,506	18,260	56	67,949	39,054
ms	Q1	5	5,075	2,909	8	7,266	4,092	11	12,774	7,391	23	27,646	15,891
	median	5	5,139	2,947	8	7,353	4,157	11	12,994	7,504	23	28,082	16,085
	Q3	5	5,225	2,983	8	7,475	4,208	11	13,127	7,608	23	28,311	16,272

As a final remark, Table 4 shows the potential room for improvement with current SIDH-sign development. There is a considerable difference with the running time of our implementation of SIDH when compared to the up-to-date Microsoft version SIDH 3.5 [ACC+20], available at https://github.com/microsoft/PQCrypto-SIDH. This is mainly due to our prime field arithmetic implementation being heavily based on CSIDH. Hence, resorting to different methods that are applied to the Microsoft implementation could possibly surpass our current performance achievements for SIDH-sign. For example, a possible speed up could be the use of lazy reductions to avoid unnecessary computations over the prime field arithmetic. We emphasize that any improvement in this direction can be highly effective to SIDH-sign, as isogeny computations are performed during signature and verification over κ iterations. Observe that our current implementation of SIDH runs at least 1.16 times slower than its counterpart for p610, while p434, p503 and p751 run approximately 1.5, 1.4 and 2.1 times slower, respectively.

On the other hand, to better understand the performance of Algorithm 4 (canonical basis), Algorithm 5 (two dimensional discrete logarithm), and [CS18, Algorithm 9] (two

Table 4: Timings measured in average of *thousands of cycles* for our SIDH implementation compared to the Microsoft version.

	p434		p503		p610		p751	
	This work	[ACC+20]	This work	[ACC+20]	This work	[ACC+20]	This work	[ACC+20]
Alice keygen	7,548	4,903	10,609	7,476	19,922	16,744	56,600	26,709
Bob keygen	8,312	5,428	11,614	8,250	21,380	16,716	62,855	30,035
Alice key derivation	6,056	3,960	8,581	6,076	16,163	13,885	46,343	21,870
Bob key derivation	6,988	4,568	9,827	6,973	18,239	14,103	53,806	25,621

dimensional ladder), Table 5 illustrates that i) Algorithm 4 is as efficient as Bob's SIDH key generation, ii) [CS18, Algorithm 9] is much cheaper than any SIDH key generation, and iii) Algorithm 5 is about 1.6x costlier than Bob's SIDH key generation. However, SIDH-sign calls (on average) $\kappa/2$ times Algorithm 5, and thus its impact becomes about the same as κ times calls of Algorithm 4. Consequently, any improvement in computing two dimensional discrete logarithms will positively impact the performance of the SIDH-sign (since it is currently the bottleneck per iteration).

Table 5: Timings measured in average of *thousands of cycles* for computations of the canonical basis (Algorithm 4), two dimensional discrete log (Algorithm 5) and two dimensional ladder ([CS18, Algorithm 9]).

	p377	p434	p503	p546	p610	p697	p751
Algorithm 4	6,027	8,498	10,791	14,468	20,450	31,256	61,509
Algorithm 5	9,150	13,144	18,314	23,790	34,694	44,570	97,361
[CS18, Algorithm 9]	1,203	1,859	2,320	3,284	4,322	5,871	11,259

5 Concluding remarks

Our results center on the weak-SIDH relation. Thus, any implementation of the PoK under the strong-SIDH relation [DDGZ21, §6], which permits SIDH-key validations, will (roughly) cost twice what we get. Similarly, using the compressed SIDH-keys to reduce the signature sizes increase the timings of SIDH-sign. We do not focus on the strong-SIDH and compressed SIDH key-based constructions, we leave them as future research.

Algorithm 4 permits an efficient free-inverse entangled basis generation with shared Elligator [PDJ21, §3, Algorithm 3.2], which has smaller precomputed tables concerning QR and QNR elements. We work with tables of 32 bytes (byte per element), while current compressed SIKE implementations use precomputed tables of 20 quadratic field elements giving $20 \times 2 \left\lfloor \frac{\log_2(p)}{8} \right\rceil$ bytes (e.g., SIKEp377 requires 1920 bytes per table). Our implementation of Algorithm 4 is as efficient as the entangled basis generation used in SIKE [ACC+20, PDJ21] but with much smaller precomputation stored.

On the other hand, it is worth highlighting that we can also use Algorithm 4 to efficiently perform the built-in block `CanonicalBasis` in the isogeny-based primitive of [FP21]. Likewise, a slight modification of Algorithm 5 allows finding 2-dimensional discrete logarithms on the 2^a-torsion subgroup $E[2^a]$, and thus we can use it as the decomposition by scalars building block of [FP21].

References

[ACC+20] Reza Azarderakhsh, Matthew Campagna, Craig Costello, Luca De Feo, Basil Hess, Amir Jalali, David Jao, Brian Koziel, Brian LaMacchia, Patrick Longa, Michael Naehrig, Geovandro Pereira, Joost Renes, Vladimir Soukharev, and David Urbanik. Supersingular Isogeny Key Encapsulation. Third Round

Candidate of the NIST's post-quantum cryptography standardization process, 2020. Available at: https://sike.org/.

[BBP+21] Jean-François Biasse, Xavier Bonnetain, Benjamin Pring, André Schrottenloher, and William Youmans. A trade-off between classical and quantum circuit size for an attack against CSIDH. *Journal of Mathematical Cryptology*, 15(1):4–17, 2021.

[BDK+21] Ward Beullens, Samuel Dobson, Shuichi Katsumata, Yi-Fu Lai, and Federico Pintore. Group Signatures and More from Isogenies and Lattices: Generic, Simple, and Efficient. *IACR Cryptol. ePrint Arch.*, page 1366, 2021.

[BKM+20] Andrea Basso, Péter Kutas, Simon-Philipp Merz, Christophe Petit, and Charlotte Weitkämper. On Adaptive Attacks Against Jao-Urbanik's Isogeny-Based Protocol. In Abderrahmane Nitaj and Amr M. Youssef, editors, *Progress in Cryptology - AFRICACRYPT 2020 - 12th International Conference on Cryptology in Africa, Cairo, Egypt, July 20-22, 2020, Proceedings*, volume 12174 of *Lecture Notes in Computer Science*, pages 195–213. Springer, 2020.

[BKV19] Ward Beullens, Thorsten Kleinjung, and Frederik Vercauteren. CSI-FiSh: Efficient Isogeny Based Signatures Through Class Group Computations. In Steven D. Galbraith and Shiho Moriai, editors, *Advances in Cryptology - ASIACRYPT 2019 - 25th International Conference on the Theory and Application of Cryptology and Information Security, Kobe, Japan, December 8-12, 2019, Proceedings, Part I*, volume 11921 of *Lecture Notes in Computer Science*, pages 227–247. Springer, 2019.

[BPW12] David Bernhard, Olivier Pereira, and Bogdan Warinschi. How Not to Prove Yourself: Pitfalls of the Fiat-Shamir Heuristic and Applications to Helios. In Xiaoyun Wang and Kazue Sako, editors, *Advances in Cryptology - ASIACRYPT 2012 - 18th International Conference on the Theory and Application of Cryptology and Information Security, Beijing, China, December 2-6, 2012. Proceedings*, volume 7658 of *Lecture Notes in Computer Science*, pages 626–643. Springer, 2012.

[CCC+19] Daniel Cervantes-Vázquez, Mathilde Chenu, Jesús-Javier Chi-Domínguez, Luca De Feo, Francisco Rodríguez-Henríquez, and Benjamin Smith. Stronger and Faster Side-Channel Protections for CSIDH. In Peter Schwabe and Nicolas Thériault, editors, *Progress in Cryptology - LATINCRYPT 2019 - 6th International Conference on Cryptology and Information Security in Latin America, Santiago de Chile, Chile, October 2-4, 2019, Proceedings*, volume 11774 of *Lecture Notes in Computer Science*, pages 173–193. Springer, 2019.

[CH17] Craig Costello and Hüseyin Hisil. A Simple and Compact Algorithm for SIDH with Arbitrary Degree Isogenies. In Tsuyoshi Takagi and Thomas Peyrin, editors, *Advances in Cryptology - ASIACRYPT 2017 Part II*, volume 10625 of *LNCS*, pages 303–329. Springer, 2017.

[CJL+17] Craig Costello, David Jao, Patrick Longa, Michael Naehrig, Joost Renes, and David Urbanik. Efficient Compression of SIDH Public Keys. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *Advances in Cryptology - EUROCRYPT 2017 - 36th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Paris, France, April 30 - May 4, 2017, Proceedings, Part I*, volume 10210 of *Lecture Notes in Computer Science*, pages 679–706, 2017.

[CLM+18] Wouter Castryck, Tanja Lange, Chloe Martindale, Lorenz Panny, and Joost Renes. CSIDH: An Efficient Post-Quantum Commutative Group Action. In Thomas Peyrin and Steven D. Galbraith, editors, *Advances in Cryptology - ASIACRYPT 2018 - 24th International Conference on the Theory and Application of Cryptology and Information Security, Brisbane, QLD, Australia, December 2-6, 2018, Proceedings, Part III*, volume 11274 of *Lecture Notes in Computer Science*, pages 395–427. Springer, 2018.

[CS18] Craig Costello and Benjamin Smith. Montgomery curves and their arithmetic - The case of large characteristic fields. *Journal of Cryptographic Engineering*, 8(3):227–240, 2018.

[CSCJR21] Jorge Chávez-Saab, Jesús-Javier Chi-Domínguez, Samuel Jaques, and Francisco Rodríguez-Henríquez. The SQALE of CSIDH: sublinear Vélu quantum-resistant isogeny action with low exponents. *Journal of Cryptographic Engineering*, 2021.

[DDGZ21] Luca De Feo, Samuel Dobson, Steven D. Galbraith, and Lukas Zobernig. SIDH Proof of Knowledge. *IACR Cryptol. ePrint Arch.*, page 1023, 2021.

[DG19] Luca De Feo and Steven D. Galbraith. SeaSign: Compact Isogeny Signatures from Class Group Actions. In Yuval Ishai and Vincent Rijmen, editors, *Advances in Cryptology - EUROCRYPT 2019 - 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, May 19-23, 2019, Proceedings, Part III*, volume 11478 of *Lecture Notes in Computer Science*, pages 759–789. Springer, 2019.

[DGL+20] Samuel Dobson, Steven D. Galbraith, Jason T. LeGrow, Yan Bo Ti, and Lukas Zobernig. An adaptive attack on 2-SIDH. *International Journal of Computer Mathematics: Computer Systems Theory*, 5(4):282–299, 2020.

[DJP14] Luca De Feo, David Jao, and Jérôme Plût. Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies. *Journal of Mathematical Cryptology*, 8(3):209–247, 2014.

[DKL+20] Luca De Feo, David Kohel, Antonin Leroux, Christophe Petit, and Benjamin Wesolowski. SQISign: Compact Post-quantum Signatures from Quaternions and Isogenies. In Shiho Moriai and Huaxiong Wang, editors, *Advances in Cryptology - ASIACRYPT 2020 - 26th International Conference on the Theory and Application of Cryptology and Information Security, Daejeon, South Korea, December 7-11, 2020, Proceedings, Part I*, volume 12491 of *Lecture Notes in Computer Science*, pages 64–93. Springer, 2020.

[DLW22] Luca De Feo, Antonin Leroux, and Benjamin Wesolowski. New algorithms for the Deuring correspondence: SQISign twice as fast. *IACR Cryptol. ePrint Arch.*, page 234, 2022.

[DPV19] Thomas Decru, Lorenz Panny, and Frederik Vercauteren. Faster SeaSign Signatures Through Improved Rejection Sampling. In Jintai Ding and Rainer Steinwandt, editors, *Post-Quantum Cryptography - 10th International Conference, PQCrypto 2019, Chongqing, China, May 8-10, 2019 Revised Selected Papers*, volume 11505 of *Lecture Notes in Computer Science*, pages 271–285. Springer, 2019.

[EKA22] Rami Elkhatib, Brian Koziel, and Reza Azarderakhsh. Faster Isogenies for Post-quantum Cryptography: SIKE. In Steven D. Galbraith, editor, *Topics*

in Cryptology - CT-RSA 2022 - Cryptographers' Track at the RSA Conference 2022, Virtual Event, March 1-2, 2022, Proceedings, volume 13161 of *Lecture Notes in Computer Science*, pages 49–72. Springer, 2022.

[EKP20] Ali El Kaafarani, Shuichi Katsumata, and Federico Pintore. Lossy CSI-FiSh: Efficient Signature Scheme with Tight Reduction to Decisional CSIDH-512. In Aggelos Kiayias, Markulf Kohlweiss, Petros Wallden, and Vassilis Zikas, editors, *Public-Key Cryptography - PKC 2020 - 23rd IACR International Conference on Practice and Theory of Public-Key Cryptography, Edinburgh, UK, May 4-7, 2020, Proceedings, Part II*, volume 12111 of *Lecture Notes in Computer Science*, pages 157–186. Springer, 2020.

[FP21] Tako Boris Fouotsa and Christophe Petit. SHealS and HealS: Isogeny-Based PKEs from a Key Validation Method for SIDH. In Mehdi Tibouchi and Huaxiong Wang, editors, *Advances in Cryptology - ASIACRYPT 2021 - 27th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 6-10, 2021, Proceedings, Part IV*, volume 13093 of *Lecture Notes in Computer Science*, pages 279–307. Springer, 2021.

[FP22] Tako Boris Fouotsa and Christophe Petit. A New Adaptive Attack on SIDH. In Steven D. Galbraith, editor, *Topics in Cryptology - CT-RSA 2022 - Cryptographers' Track at the RSA Conference 2022, Virtual Event, March 1-2, 2022, Proceedings*, volume 13161 of *Lecture Notes in Computer Science*, pages 322–344. Springer, 2022.

[FS86] Amos Fiat and Adi Shamir. How to Prove Yourself: Practical Solutions to Identification and Signature Problems. In Andrew M. Odlyzko, editor, *Advances in Cryptology - CRYPTO '86, Santa Barbara, California, USA, 1986, Proceedings*, volume 263 of *Lecture Notes in Computer Science*, pages 186–194. Springer, 1986.

[GPS20] Steven D. Galbraith, Christophe Petit, and Javier Silva. Identification Protocols and Signature Schemes Based on Supersingular Isogeny Problems. *Journal of Cryptology*, 33(1):130–175, 2020.

[GPST16] Steven D. Galbraith, Christophe Petit, Barak Shani, and Yan Bo Ti. On the Security of Supersingular Isogeny Cryptosystems. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *Advances in Cryptology - ASIACRYPT 2016 - 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part I*, volume 10031 of *Lecture Notes in Computer Science*, pages 63–91, 2016.

[JD11] David Jao and Luca De Feo. Towards Quantum-Resistant Cryptosystems from Supersingular Elliptic Curve Isogenies. In Bo-Yin Yang, editor, *Post-Quantum Cryptography - 4th International Workshop, PQCrypto 2011, Taipei, Taiwan, November 29 - December 2, 2011. Proceedings*, volume 7071 of *Lecture Notes in Computer Science*, pages 19–34. Springer, 2011.

[Lon22] Patrick Longa. Efficient Algorithms for Large Prime Characteristic Fields and Their Application to Bilinear Pairings and Supersingular Isogeny-Based Protocols. *IACR Cryptol. ePrint Arch.*, page 367, 2022.

[LWS21] Patrick Longa, Wen Wang, and Jakub Szefer. The Cost to Break SIKE: A Comparative Hardware-Based Analysis with AES and SHA-3. In Tal Malkin and Chris Peikert, editors, *Advances in Cryptology - CRYPTO 2021 - 41st*

Annual International Cryptology Conference, CRYPTO 2021, Virtual Event, August 16-20, 2021, Proceedings, Part III, volume 12827 of *Lecture Notes in Computer Science*, pages 402–431. Springer, 2021.

[LZ19] Qipeng Liu and Mark Zhandry. Revisiting Post-quantum Fiat-Shamir. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology - CRYPTO 2019 - 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2019, Proceedings, Part II*, volume 11693 of *Lecture Notes in Computer Science*, pages 326–355. Springer, 2019.

[PB21] Geovandro C. C. F. Pereira and Paulo S. L. M. Barreto. Isogeny-Based Key Compression Without Pairings. In Juan A. Garay, editor, *Public-Key Cryptography - PKC 2021 - 24th IACR International Conference on Practice and Theory of Public Key Cryptography, Virtual Event, May 10-13, 2021, Proceedings, Part I*, volume 12710 of *Lecture Notes in Computer Science*, pages 131–154. Springer, 2021.

[PDJ21] Geovandro C. C. F. Pereira, Javad Doliskani, and David Jao. x-only point addition formula and faster compressed SIKE. *Journal of Cryptographic Engineering*, 11(1):57–69, 2021.

[Pei20] Chris Peikert. He Gives C-Sieves on the CSIDH. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology - EUROCRYPT 2020 - 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10-14, 2020, Proceedings, Part II*, volume 12106 of *Lecture Notes in Computer Science*, pages 463–492. Springer, 2020.

[Ren18] Joost Renes. Computing Isogenies Between Montgomery Curves Using the Action of (0, 0). In Tanja Lange and Rainer Steinwandt, editors, *Post-Quantum Cryptography - 9th International Conference, PQCrypto 2018, Fort Lauderdale, FL, USA, April 9-11, 2018, Proceedings*, volume 10786 of *LNCS*, pages 229–247. Springer, 2018.

[Sto12] Anton Stolbunov. *Cryptographic Schemes Based on Isogenies*. PhD thesis, Norwegian University of Science and Technology Faculty of Information Technology, Mathematics and Electrical Engineering Department of Telematics, 01 2012.

[Unr17] Dominique Unruh. Post-quantum Security of Fiat-Shamir. In *International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT)*, 2017.

[YAJ+17] Youngho Yoo, Reza Azarderakhsh, Amir Jalali, David Jao, and Vladimir Soukharev. A Post-quantum Digital Signature Scheme Based on Supersingular Isogenies. In Aggelos Kiayias, editor, *Financial Cryptography and Data Security - 21st International Conference, FC 2017, Sliema, Malta, April 3-7, 2017, Revised Selected Papers*, volume 10322 of *Lecture Notes in Computer Science*, pages 163–181. Springer, 2017.