SQIsign

Algorithm specifications and supporting documentation

Version 1.0

- Jorge Chavez-Saab, *Technology Innovation Institute, UAE*
- Maria Corte-Real Santos, *University College London, UK*
- Luca De Feo, *IBM Research Europe, Switzerland*
- Jonathan Komada Eriksen, *Norwegian University of Science and Technology, Norway*
- Basil Hess, *IBM Research Europe, Switzerland*
- David Kohel, *Institut de Mathématiques de Marseille, Aix-Marseille University, France*
- Antonin Leroux, *DGA-MI, Bruz, France and Université de Rennes, France*
- Patrick Longa, *Microsoft Research, USA*
- Michael Meyer, *University of Regensburg, Germany*
- Lorenz Panny, *Academia Sinica, Taiwan*
- Sikhar Patranabis, *IBM Research India*
- Christophe Petit, *Université libre de Bruxelles, Belgium and University of Birmingham, UK*
- Francisco Rodríguez Henríquez, *Technology Innovation Institute, UAE*
- Sina Schaeffler, *IBM Research Europe and ETH Zürich, Switzerland*
- Benjamin Wesolowski, *CNRS and ENS de Lyon, France*

June 1, 2023

https://sqisign.org

Contents

CHAPTER 1

Introduction

This document presents a detailed description of the digital signature scheme SQIsign, whose security is based on the presumed hardness of finding isogenies between supersingular elliptic curves. The scheme is based on the original construction of [DKL+20], and includes several subsequent improvements.

Compact keys and signatures. SQIsign offers both very compact public keys and signatures (see Table 1). To our knowledge, SQIsign has the smallest combined size of public key and signature of any post-quantum signature scheme.

Confident tuning of security parameters. The complexity of the fastest known attacks against SQIsign is well understood. It allows for straightforward, precise estimation of concrete attack costs, hence confident tuning of security parameters. While this complexity has been very stable, it is worth recalling that the history of the scheme is short. The best known attacks amount to solving the *endomorphism ring problem*[1]; while arithmeticians were already looking at this computational problem thirty years ago, it has only recently received the spotlight of cryptography.

Relatively simple and fast verification. The verification procedure is reasonably efficient and simple to implement, involving mostly elementary elliptic curve arithmetic, and the computation of low-degree isogeny chains *à la* SIKE.

A complex signing procedure. The main drawback of SQIsign is the intricacy of the signing procedure, in terms of mathematical sophistication and diversity of objects being manipulated. This renders the signature relatively difficult to implement, and signing is much slower than competing post-quantum schemes.

The present specifications of SQIsign start in Chapter 2 with an overview of the different mathematical objects used in SQIsign, and a precise description of the algorithms to manipulate them. Then, Chapter 3 details the components of SQIsign itself (generating keys, signing, and verifying), and provides the parameter sets. The following chapters discuss the choice of parameters, performance, and security.

1.1. High level description of SQIsign

SQIsign is designed as a proof of knowledge (a sigma protocol), turned into a signature by the Fiat-Shamir transform. The sigma protocol proves knowledge of an *elliptic curve endomorphism*.

[1] Note that the endomorphism ring problem, hence the security of SQIsign, is not affected by the polynomial-time attacks [CD23, MMP+23, Rob23] against the SIDH [JD11] key exchange. While SIDH also belonged to the "isogeny-based" family, it relied on a weaker variant of the fundamental isogeny problems.

Table 1. SQIsign key and signature sizes in bytes for each security level.

Parameter set	`Public key`	`Secret key`	`Signature`
NIST-I	64	782	177
NIST-III	96	1138	263
NIST-V	128	1509	335

We consider a collection of objects called *elliptic curves*. Two elliptic curves may be connected by maps called *isogenies*. We write $\varphi : E_1 \to E_2$ for an isogeny φ connecting an elliptic curve E_1 to an elliptic curve E_2. The foundational problem of isogeny-based cryptography is essentially the following *isogeny path problem*: given two elliptic curves E_1 and E_2, find an isogeny $E_1 \to E_2$.

Now, given an elliptic curve E, an *endomorphism of* E is an isogeny $\varphi : E \to E$. The collection of all endomorphisms of E is called the *endomorphism ring* of E, written $\mathrm{End}(E)$. The *endomorphism ring problem* is the following: given E, compute $\mathrm{End}(E)$. For so-called *supersingular* elliptic curves, this problem is known to be equivalent to the isogeny path problem under polynomial time reductions [**EHL+18**, **Wes22**]. In fact, we have that

- given E_1, E_2, $\mathrm{End}(E_1)$ and $\mathrm{End}(E_2)$, one can find an isogeny $E_1 \to E_2$ in polynomial time, and
- given E_1, E_2, $\mathrm{End}(E_1)$, and an isogeny $E_1 \to E_2$, one can compute $\mathrm{End}(E_2)$ in polynomial time.

The sigma protocol now works as follows. The prover has as public key an elliptic curve E_A, and their secret is the associated endomorphism ring $\mathrm{End}(E_A)$. With E_A public, the goal of the prover is to convince the verifier that they know $\mathrm{End}(E_A)$. They proceed as follows.

(1) For the commitment phase, the prover generates a random pair $(E_1, \mathrm{End}(E_1))$, and sends E_1 to the verifier.
(2) For the challenge phase, the verifier generates a random isogeny $\varphi_{\text{chall}} : E_1 \to E_2$ and sends it to the prover.
(3) Given $\mathrm{End}(E_1)$ and $\varphi_{\text{chall}} : E_1 \to E_2$, the prover can compute $\mathrm{End}(E_2)$. Now, given the secret $\mathrm{End}(E_A)$ and the freshly computed $\mathrm{End}(E_2)$, the prover can compute an isogeny $\varphi_{\text{resp}} : E_A \to E_2$, and send it to the verifier.
(4) The verifier checks that φ_{resp} is indeed an isogeny from the public key E_A to the challenge curve E_2.

The idea is that to compute the response, the prover must use their knowledge of $\mathrm{End}(E_A)$. Intuitively, the protocol asks for the prover to compute isogenies from E_A to a somewhat random curve E_2, and that relates to knowledge of $\mathrm{End}(E_A)$ thanks to the computational equivalence between the isogeny path problem and the endomorphism ring problem. However, this idea does not immediately work, and the protocol, as sketched above, is insecure: a cheating prover could generate the commitment E_1 by choosing a random isogeny $\varphi_{\text{com}}^{\text{cheat}} : E_A \to E_1$. They may not be able to compute $\mathrm{End}(E_1)$, but it does not matter. In response to $\varphi_{\text{chall}} : E_1 \to E_2$, they would simply respond with $\varphi_{\text{resp}}^{\text{cheat}} = \varphi_{\text{chall}} \circ \varphi_{\text{com}}^{\text{cheat}} : E_A \to E_2$.

There is a simple fix to this issue, by ensuring that φ_{chall} is not a "sub-isogeny" of φ_{resp}. With this fix, one can actually prove that this protocol proves knowledge of *at least* some non-trivial part of $\mathrm{End}(E_A)$, which is believed to be as hard as knowing the full $\mathrm{End}(E_A)$.

The SQISIGN protocol described in detail in Chapter 3 follows the above outline. There is a notable cosmetic difference: we fix a public reference pair $(E_0, \mathrm{End}(E_0))$, and instead of thinking of the keys as a random pair $(E_A, \mathrm{End}(E_A))$, we think of them as a random isogeny $\varphi_{\text{secret}} : E_0 \to E_A$. Similarly, we generate the commitment via a random isogeny $\varphi_{\text{com}} : E_0 \to E_1$ instead of a random pair $(E_1, \mathrm{End}(E_1))$. Both approaches are computationally equivalent, but in order to generate a random pair $(E, \mathrm{End}(E))$, one would typically start by computing a random isogeny $E_0 \to E$.

1.2. Differences with the original SQISIGN article

The main difference between the present version of SQISIGN and the original publication [**DKL+20**] is the inclusion of the improvements described in [**DLLW23**]. In addition, we propose new parameter sets for the security levels NIST-I, NIST-III, and NIST-V. Finally, the size of the signatures has been further reduced by improving the compression and decompression algorithms. Note that compared to [**DKL+20**, **DLLW23**], the code has been written from scratch (without dependences to PARI/GP), and some of the finest optimizations are not yet present in the reference implementation. Future optimizations are expected to reach at least the performance of [**DLLW23**].

CHAPTER 2

Basic operations

SQIsign is based on a mathematical correspondence between two seemingly distant mathematical worlds: supersingular **elliptic curves** and **isogenies** defined over **finite fields** on one side, **maximal orders** and **ideals** of **quaternion algebras** on the other side. A complete implementation of SQIsign must be capable of representing all these objects and manipulating them. We describe finite fields, elliptic curves and isogenies in Sections 2.1–2.3; this is as much as is needed for verification. Sections 2.4 and 2.5 describe quaternion algebras and related concepts; Section 2.6 describes algorithms for moving between the world of quaternions and that of elliptic curves. They are all needed for key-generation and signature.

2.1. Finite fields

We follow the presentation in [JAC+20]. A finite field is a finite set of elements equipped with an addition and multiplication operation. In particular, addition and multiplication are closed, there exist additive resp. multiplicative neutral elements 0 resp. 1, and additive resp. multiplicative inverses of each element, with the exclusion of the only non-multiplicatively-invertible element 0.

Finite fields of cardinality q exist if and only if q is a prime power, i.e., $q = p^r$ for some prime number p and positive integer r. Such finite fields of cardinality q have a unique representation up to isomorphism, and are denoted by $\mathbb{F}_q$. For $q = p^r$, we call $\mathrm{char}(\mathbb{F}_q) = p$ the characteristic of $\mathbb{F}_q$. SQIsign uses fields of special characteristic p, satisfying $p \equiv 3 \pmod 4$ as detailed in Chapter 5.

2.1.1. The finite field $\mathbb{F}_p$

We uniquely represent the elements of the finite field $\mathbb{F}_p$ by the integers $\{0, \ldots, p-1\}$. The algebraic operations are defined as follows:

Addition: For $a, b \in \mathbb{F}_p$, the sum $c = a + b$ is given by the unique integer $c \in \{0, \ldots, p-1\}$ that satisfies $c \equiv a + b \pmod p$.

Additive inverse: For $a \in \mathbb{F}_p$, its additive inverse $-a$ is given by the unique integer $(-a) \in \{0, \ldots, p-1\}$ satisfying $a + (-a) \equiv 0 \pmod p$.

Multiplication: For $a, b \in \mathbb{F}_p$, the product $c = a \cdot b$ is given by the unique integer $c \in \{0, \ldots, p-1\}$ that satisfies $c \equiv a \cdot b \pmod p$.

Multiplicative inverse: For $a \in \mathbb{F}_p$, $a \neq 0$, its multiplicative inverse a^{-1} is given by the unique integer $a^{-1} \in \{0, \ldots, p-1\}$ satisfying $a \cdot a^{-1} \equiv 1 \pmod p$.

Quadratic residuosity: Let $a \in \mathbb{F}_p$, decide whether a is a square, i.e., whether there is an element $b \in \mathbb{F}_p$ with $b^2 = a$. This is done by computing the Legendre symbol $a^{\frac{p-1}{2}}$, which equals 1 if a is a square, -1 otherwise.

Square root: Let $a \in \mathbb{F}_p$ be a square in $\mathbb{F}_p$. Since we restrict to primes satisfying $p \equiv 3 \pmod 4$, we compute the canonical square root of a as

$$\sqrt{a} = a^{\frac{p+1}{4}} \pmod p. \tag{1}$$

Additionally, we define an **ordering** on elements of $\mathbb{F}_p$ by lifting them to the interval $[0, p-1]$ and comparing integers.

2.1.2. The finite field $\mathbb{F}_{p^2}$

Since we will only use finite fields of characteristic $p \equiv 3 \pmod 4$, we can define the field extension $\mathbb{F}_{p^2}$ as $\mathbb{F}_{p^2} = \mathbb{F}_p(i)$ with $i^2 + 1 = 0$. We uniquely represent the elements of $\mathbb{F}_{p^2}$ as $a = a_0 + a_1 \cdot i$ with $a_0, a_1 \in \mathbb{F}_p$. The algebraic operations are defined as follows:

Addition: For $a, b \in \mathbb{F}_{p^2}$, their sum is given by $c = c_0 + c_i \cdot i$ with $c_0 = a_0 + b_0$ and $c_1 = a_1 + b_1$, using additions in $\mathbb{F}_p$.

Additive inverse: For $a \in \mathbb{F}_{p^2}$, its additive inverse $-a$ is given by $-a = (-a_0) + (-a_1) \cdot i$, using additive inversions in $\mathbb{F}_p$.

Multiplication: For $a, b \in \mathbb{F}_{p^2}$, their product is given by $c = c_0 + c_i \cdot i$ with $c_0 = a_0 b_0 - a_1 b_1$ and $c_1 = a_0 b_1 + a_1 b_0$, using additions, additive inversions, and multiplications in $\mathbb{F}_p$.

Multiplicative inverse: For $a \in \mathbb{F}_{p^2}$, $a \neq 0$, its multiplicative inverse is given by $a^{-1} = (a_0 N^{-1}) + (-a_1 N^{-1}) \cdot i$, where $N = a_0^2 + a_1^2 \in \mathbb{F}_p$, using additions, additive inversions, multiplications and multiplicative inversions in $\mathbb{F}_p$.

Quadratic residuosity: Let $a \in \mathbb{F}_{p^2}$, decide whether a is a square. This is the case if and only if $a^{p+1} = a_0^2 + a_1^2 \in \mathbb{F}_p$ is a square in $\mathbb{F}_p$.

Square root: Let $a \in \mathbb{F}_p$. If a is a square in $\mathbb{F}_p$, we define its square root as in Eq. (1); otherwise $-a$ is a square in $\mathbb{F}_p$ and we define $\sqrt{a} = \sqrt{-a} \cdot i$. Finally, let $a \in \mathbb{F}_{p^2} \setminus \mathbb{F}_p$ be a square in $\mathbb{F}_{p^2}$, i.e., there is an element $b \in \mathbb{F}_{p^2}$ with $b^2 = a$. We use [**ARH14**, Algorithm 8] to define a canonical square root of $a = a_0 + a_1 \cdot i$ as $\sqrt{a} = x_0 + x_1 \cdot i$, where

$$x_0 = \sqrt{\frac{a_0 \pm \sqrt{a_0^2 + a_1^2}}{2}}, \qquad x_1 = \frac{a_1}{2x_0}, \tag{2}$$

knowing that only one of $\left(a_0 + \sqrt{a_0^2 + a_1^2}\right)/2$ and $\left(a_0 - \sqrt{a_0^2 + a_1^2}\right)/2$ will have a canonical square root in $\mathbb{F}_p$.

Additionally, we define a **lexicographic ordering** on elements of $\mathbb{F}_{p^2}$ by

$$a_0 + a_1 \cdot i \ < \ b_0 + b_1 \cdot i \quad \text{iff} \quad a_0 < b_0 \text{ or } (a_0 = b_0 \text{ and } a_1 < b_1). \tag{3}$$

2.2. Elliptic curves

In the following, we assume that $\mathbb{F}_q$ is a finite field with $\mathrm{char}(\mathbb{F}_q) > 2$. We recall here some key facts on supersingular elliptic curves in Montgomery form necessary to the implementation of SQIsign. For a an extensive review of Montgomery curves and their properties, see [**CS18**].

2.2.1. Montgomery curves

Let $A, B \in \mathbb{F}_q$ such that $B(A^2 - 4) \neq 0$. The Montgomery curve $E_{A,B}$ over $\mathbb{F}_q$ is an elliptic curve defined by the equation

$$By^2 = x^3 + Ax^2 + x. \tag{4}$$

That is, it consists of the set of points $P = (x, y)$ that satisfy the curve equation (for x and y in any extension of the field $\mathbb{F}_q$), and the point at infinity ∞. We often write $E_{A,B}/\mathbb{F}_q$ to emphasize that the curve is defined over the field $\mathbb{F}_q$. We also write E_A when $B = 1$, and we write E for a generic Montgomery curve. Furthermore, we write $E_{A,B}(\mathbb{F}_q)$ to denote the set of points of $E_{A,B}$ (defined over $\mathbb{F}_q$). The coefficient A is often called a *Montgomery A-invariant*.

Two Montgomery curves are said to be *isomorphic* over $\mathbb{F}_q$ if there is a linear change of coordinates $(x, y) \mapsto (D(x + R), Cy)$, with $D, C, R \in \mathbb{F}_q$ that maps one onto the other. When $E_{A,B}$ and $E_{A',B'}$ are not isomorphic over $\mathbb{F}_q$, but they are over a finite extension of $\mathbb{F}_q$, we say that they are the *twist* of one another. In particular, $E_{A,B}$ and $E_{A,B'}$ are always the *quadratic* twist of one another by taking $C = \sqrt{B/B'}$, and are isomorphic if and only if B/B' is a square in $\mathbb{F}_q$.

Let N be the number of solutions to Equation (4) (including the point ∞). When $N = 1 \mod \mathrm{char}(\mathbb{F}_q)$, we say that $E_{A,B}$ is *supersingular*. We are only interested in supersingular curves defined over $\mathbb{F}_{p^2}$ with $p = 3 \bmod 4$. In this case, any supersingular curve $E_A/\mathbb{F}_{p^2}$ (with $B = 1$) has exactly $(p+1)^2$ points, whereas its quadratic twists $E_{A,\gamma}/\mathbb{F}_{p^2}$, where γ is an arbitrary quadratic non-residue in $\mathbb{F}_{p^2}$, have exactly $(p-1)^2$ points and are all isomorphic. SQIsign always manipulates pairs $(E_A, E_{A,\gamma})$ of supersingular curves with a quadratic twist, where γ can be taken as a fixed non-residue across all curves.[1]

2.2.1.1. Isomorphism class and canonical representative. For a Montgomery curve $E_{A,B}$ we define its j-invariant

$$j(E_{A,B}) = \frac{256(A^2-3)^3}{A^2-4}. \tag{5}$$

The j-invariant characterizes isomorphism classes of elliptic curves over the algebraic closure, i.e., two curves have the same j-invariant if and only if they are isomorphic or are twists of one another. To a pair $(E_A, E_{A,\gamma})$, SQIsign associates their common j-invariant $j(E_A)$ (note that j does not depend on B) as a unique identifier for the isomorphism class of the pair.

To a single supersingular j-invariant are associated six Montgomery A-invariants, defining isomorphic curves. Among these six isomorphic choices some of SQIsign's routines need to choose one in a way that only depends on the isomorphism class. This is done using Algorithm MontgomeryNormalize below.

Algorithm 1 MontgomeryNormalize(A)

Input: A Montgomery A-invariant A
Output: A *normalized* Montgomery A-invariant A' in the same isomorphism class, an isomorphism from E_A to $E_{A'}$.

1: Let
$$Z_0 := A^2, \qquad Z_1 := \frac{9-A^2}{2} + \frac{A^3-3A}{2\sqrt{A^2-4}}, \qquad Z_2 := \frac{9-A^2}{2} - \frac{A^3-3A}{2\sqrt{A^2-4}};$$
2: Let $Z = z_0 + z_1 \cdot i$ be the lexicographically smallest (see Eq. (3)) among Z_0, Z_1, Z_2;
3: $A' := \sqrt{Z}$;
4: **if** $A' = A$ **then**
5: $\quad R := 0, \qquad U := 1;$
6: **else if** $A' = -A$ **then**
7: $\quad R := 0, \qquad U := i;$
8: **else**
9: $\quad R := \frac{(A^2+A'^2-6)A}{A^2+2A'^2-9};$
10: $\quad U := \sqrt{\frac{A'}{A-3R}};$
11: **end if**
12: **return** A' and the isomorphism $(x, y) \mapsto (U^2(x+R), U^3 y)$.

2.2.2. The group law

The set of points of a Montgomery curve, equipped with the addition operation defined in Sections 2.2.2.1 and 2.2.2.2, forms an abelian group. Under this addition law, each point $P = (x, y)$ has an inverse $-P = (x, -y)$, and ∞ is the identity element.

In what follows, for a point $P \neq \infty$, we refer to its x-coordinate as x_P, and to its y-coordinate as y_P, i.e., $P = (x_P, y_P)$. Note that optimized implementations typically use projective coordinates $(X : Y : Z)$ with $x = X/Z$ and $y = Y/Z$ in order to avoid inversions in the point addition and isogeny formulas below (see, e.g., [**CS18**]).

[1] Optimized implementations may ignore γ and just store A.

2.2.2.1. Point addition. Let $E_{A,B}/\mathbb{F}_q$ be a Montgomery curve, and $P = (x_P, y_P)$ and $Q = (x_Q, y_Q)$ points on $E_{A,B}$ with $P \neq \pm Q$. Then we compute their sum $R = P + Q$ with $R = (x_R, y_R)$ as

$$x_R = B\lambda^2 - (x_P + x_Q) - A,$$

and

$$y_R = \lambda(x_P - x_R) - y_P,$$

where $\lambda = (y_P - y_Q)/(x_P - x_Q)$.

The point at infinity is the neutral element of the law, so $P + \infty = \infty + P = P$.

2.2.2.2. Point doubling. Let $E_{A,B}/\mathbb{F}_q$ be a Montgomery curve, and $P = (x_P, y_P)$ a point on $E_{A,B}$ with $P \neq -P$. We compute its double $[2]P = P + P = (x_{[2]P}, y_{[2]P})$ as

$$x_{[2]P} = \frac{(x_P^2 - 1)^2}{4x_P(x_P^2 + Ax_P + 1)},$$

and

$$y_{[2]P} = y_P \cdot \frac{(x_P^2 - 1)(x_P^4 + 2Ax_P^3 + 6x_P^2 + Ax_p + 1)}{8x_P^2(x_P^2 + Ax_P + 1)^2}.$$

If $P = -P$, we set $[2]P = \infty$.

2.2.2.3. Scalar multiplication. Using the addition and doubling operations defined above, we can define a scalar multiplication $[k] : E \to E$ for $k \in \mathbb{Z}$: For positive k, it maps a point $P \in E$ to the point $[k]P = P + P + \cdots + P$, summing k copies of P. For negative k, we set $[k]P = -[|k|]P$. For $k = 0$, we set $[0]P = \infty$.

For efficiency, a scalar multiplication is usually performed as a sequence of point doublings and point additions. Using the Montgomery ladder (see, e.g., **[CS18]**), the number of elliptic curve point operations is logarithmic in k.

For a point $P \in E$, we call the smallest positive integer m such that $[m]P = \infty$ the order of P.

2.2.3. Torsion subgroups and deterministic basis computation

For $m \in \mathbb{Z}$ and $E/\mathbb{F}_{p^2}$ a supersingular curve, we define $E[m]$ to be the m-torsion subgroup of E, which contains all points $P \in E(\mathbb{F}_{p^2})$ such that $[m]P = \infty$. For $m^2 | \#E(\mathbb{F}_{p^2})$, we have $E[m] \cong \mathbb{Z}/m\mathbb{Z} \times \mathbb{Z}/m\mathbb{Z}$, and thus $E[m]$ has cardinality m^2. Furthermore, there are non-unique points $R, S \in E[m]$ that generate $E[m]$, i.e., $\langle R, S\rangle = E[m]$, and we call (R, S) a basis of $E[m]$.

Given $E/\mathbb{F}_{p^2}$ of order $N^2 = \#E(\mathbb{F}_{p^2})$ (recall N is either $p+1$ or $p-1$), SQISIGN needs to generate bases (R, S) of $E[m]$ for some $m|N$. The algorithm essentially takes arbitrary R and S of order m until a pair defining a basis is found. In some other cases, SQISIGN already has a point R of order m and needs a second point S that completes it to a basis of $E[m]$. These algorithms are described in detail in TorsionBasis and CompleteBasis below.[2]

2.2.4. Discrete logarithms

Given a point P and a multiple $[k]P$ on an elliptic curve E for an unknown $k \in \mathbb{Z}$, the discrete logarithm problem (DLP) asks to recover k. For points of large prime order, this problem is suspected to be hard for classical computers, and underlies the security of traditional elliptic curve cryptography like Elliptic Curve Diffie–Hellman (ECDH). However, for points P of extremely smooth order, this problem can be efficiently solved.

In particular, in SQISIGN we consider points $P \in E$ of order ℓ^f for $\ell \in \{2, 3\}$ and some $f > 0$. Given a basis (R, S) of $E[\ell^f]$, we can write $P = [a]R + [b]S$ for some $a, b \in \mathbb{Z}$ with $0 \leq a, b < \ell^f$. This, combined with the deterministic bases of $E[\ell^f]$ defined previously, lets us represent any $P \in E$ of order ℓ^f by two integers of $\lceil f \log_2(\ell) \rceil$ bits.

[2]Some implementations of SQISIGN, such as the reference implementation, may represent a basis (R, S) by a triplet (x_R, x_S, x_{RS}) of x-coordinates, for efficiency. Technically, this defines (R, S) up to a global sign, i.e., the representation is the same for (R, S) and $(-R, -S)$. These implementations may simply forego the computation of the y-coordinates in TorsionBasis and CompleteBasis; being compatible with them is the reason behind the somewhat convoluted computation of y in line 7 of CompleteBasis.

Several algorithms exist to solve discrete logarithms in $E[\ell^f]$, all tracing back to Pohlig–Hellman [**PH78**]. Here is an example of one such recursive algorithm:

(1) If $f = 1$ solve the problem by exhaustive search; otherwise
(2) Let $e = \lfloor f/2 \rfloor$;
(3) Let $R' = [\ell^{f-e}]R$, $S' = [\ell^{f-e}]S$ and $P' = [\ell^{f-e}]P$;
(4) Recursively compute w, z such that $P' = [w]R' + [z]S'$ on points of order ℓ^e;
(5) Let $R'' = [\ell^e]R$, $S'' = [\ell^e]S$ and $P'' = P - [w]R - [z]S$;
(6) Recursively compute x, y such that $P'' = [w]R'' + [z]S''$ on points of order ℓ^{f-e};

Algorithm 2 $\mathsf{CompleteBasis}_{m,N}(E_{A,B}, R, [x = 1])$

Input: A Montgomery curve $E_{A,B}$ of order N^2, with $m|N$;
Input: A point $R = (x_R, y_R) \in E_{A,B}(\mathbb{F}_{p^2})$ of order m;
Input: A starting value $x \in \mathbb{F}_{p^2}$, by default $x = 1$.
Output: A basis (R, S) of $E_{A,B}[m]$.

while True **do**
  $x := x + i$;
  if x does not define the absissa of a point on $E_{A,B}$ **then**
    Continue;
  end if
  $x_{RS} := \frac{2Ax_Rx+(x_Rx+1)(x_R+x)+\sqrt{(2Ax_Rx+(x_Rx+1)(x_R+x))^2-(x_Rx-1)^2(x_R-x)^2}}{(x_R-x)^2}$;
  $y := \frac{(x_R-x)(x_{RS}+2(x_Rx-1)(x_R+x)+4Ax_Rx)}{4y_R}$;
  $S := \big(x, y\big) \in E_{A,B}(\mathbb{F}_{p^2})$;
  $S := [N/m]P$;
  if $[m/x]S = \infty$ for any divisor $x|m$ **then**
    Continue;
  end if
  if (R, S) is a basis of $E_{A,B}[m]$ **then**
    Break;
  end if
end while
return (R, S).

Algorithm 3 $\mathsf{TorsionBasis}_{m,N}(E_{A,B})$

Input: A Montgomery curve $E_{A,B}$ with order N^2, with $m|N$
Output: A basis (R, S) of $E_{A,B}[m]$.

$x := 1$;
while True **do**
  $x := x + i$;
  if x does not define the absissa of a point on $E_{A,B}$ **then**
    Continue;
  end if
  $R := \big(x, \sqrt{(x^3 + Ax^2 + x)/B}\big) \in E_{A,B}(\mathbb{F}_{p^2})$;
  $R := [N/m]P$;
  if $[m/x]R = \infty$ for any divisor $x|m$ **then**
    Continue;
  end if
  Break;
end while
return $\mathsf{CompleteBasis}_{m,N}(E_{A,B}, R, x)$.

(7) $a = w + x\ell^e$ and $b = z + y\ell^e$ is the solution.

For compatibility with optimized implementations, SQIsign only computes the discrete logarithm up to a global sign, i.e., a pair of integers a, b such that either $P = [a]R + [b]S$ or $P = [-a]R + [-b]S$. Because of the ambiguity in the output, we define a normalized representation of the pair (a, b) up to sign in Algorithm NormalizedDlog.

Algorithm 4 $\mathsf{NormalizedDlog}_{\ell^f}(E, (R, S), P)$

Input: A basis (R, S) of $E[\ell^f]$, a point $P \in E[\ell^f]$.
Output: Normalized integers a, b such that $[a]R + [b]S = \pm P$.

```
Find integers a, b such that [a]R + [b]S = ±P using a Pohlig–Hellman-style algorithm;
if a > ⌊ℓ^f/2⌋ then
    a = ℓ^f − a;
    b = ℓ^f − b;
else if a ∈ {0, ⌈ℓ^f/2⌉} and b > ⌊ℓ^f/2⌋ then
    b = ℓ^f − b;
end if
return (a, b).
```

2.3. Isogenies

For two elliptic curves E_1 and E_2 over $\mathbb{F}_q$, an isogeny is a non-constant map $\varphi : E_1 \to E_2$ defined coordinate-wise by polynomial fractions over $\mathbb{F}_q$, that satisfies $\varphi(\infty) = \infty$. In particular, φ is a group homomorphism $\varphi : E_1 \to E_2$. Such curves E_1 and E_2 that are connected through an isogeny are called isogenous. A characterization for this property is given by the group orders: Two curves $E_1/\mathbb{F}_q$ and $E_2/\mathbb{F}_q$ are isogenous over $\mathbb{F}_q$ if and only if $\#E_1(\mathbb{F}_q) = \#E_2(\mathbb{F}_q)$.

An isogeny can be almost uniquely characterized by its kernel, i.e., the set $\ker(\varphi) = \{P \in E \mid \varphi(P) = \infty\}$. Concretely, given a subgroup $G \subset E_1$ of cardinality N, there is, up to post-composition with isomorphisms, a unique elliptic curve E_2 and isogeny $\varphi : E_1 \to E_2$ of degree N with $\ker(\varphi) = G$. Thus, given a generator Q of G, we can represent isogenies with kernel $G = \langle Q \rangle$ by a single point. Furthermore, for each isogeny $\varphi : E_1 \to E_2$ there is a unique dual isogeny $\widehat{\varphi} : E_2 \to E_1$ of the same degree N, such that the composition $\widehat{\varphi} \circ \varphi$ resp. $\varphi \circ \widehat{\varphi}$ is the scalar multiplication map $[N]$ on E_1 resp. E_2.

For the explicit computation of an isogeny φ over $\mathbb{F}_q$, we can write it as a pair of rational maps $f(x)$ and $g(x)$ over $\mathbb{F}_q$, such that $\varphi((x, y)) = (f(x), y \cdot g(x))$. We can express these functions as ratios of coprime polynomials over $\mathbb{F}_q$, e.g., $f(x) = f_1(x)/f_2(x)$, such that the degree $\deg(\varphi) = \max\{\deg(f_1), \deg(f_2)\}$.

Given a point Q of order N, Vélu's formulas [**Vél71**] provide a way to compute these rational maps for the corresponding isogeny φ with $\ker(\varphi) = \langle Q \rangle$. Vélu's formulas and variants have complexity polynomial in N, thus they are only practical for relatively small values of N. When N is large and composite, we decompose φ into smaller-degree isogenies: Let $N = \prod \ell_i^{e_i}$ be the prime factorization of N. Then we can compute φ as a composition of e_1 isogenies of degree ℓ_1, e_2 isogenies of degree ℓ_2, *et cætera*. In particular, we compute φ through $\varphi = \varphi_{\sum e_i} \circ \cdots \circ \varphi_2 \circ \varphi_1$. Since each isogeny φ_j has some prime degree $\ell_i \mid N$, this is computationally feasible if the degree N is smooth, i.e., if N only contains sufficiently small prime factors ℓ_i.

We now give explicit formulas for isogenies of small prime degree. We stress once again that the kernel only defines the isogeny up to post-composition with an isomorphism. The literature abounds in formulas for isogenies of Montgomery curves, however these may disagree on the equation of the image curve. A compliant implementation of SQIsign **must** implement the isogeny formulas given below. Failing to do so risks producing mathematically meaningful but invalid signatures.

Although we give formulas in affine (x, y) coordinates, implementations are free to use projective coordinates and/or discard the y-coordinates completely, for implementation purposes.

2.3.1. 2-isogenies

Let $Q \in E_{A,B}$ be a point of order 2 generating the kernel of a 2-isogeny $\varphi : E_{A,B} \to E_{A',B'}$. We must distinguish two cases:

$Q = (0,0)$**:** Then the isogeny is defined by

$$\varphi(x,y) = \left(\frac{1}{\sqrt{A^2-4}} \frac{x^2+Ax+1}{x}, \quad \frac{1}{\sqrt[4]{A^2-4}} \cdot y \cdot \frac{x^2-1}{x^2} \right),$$

and the image curve is defined by

$$(A',B') = \left(-\frac{2A}{\sqrt{A^2-4}}, B \right).$$

$Q = (x_Q, 0)$ **with** $x_Q \neq 0$**:** In this case we follow [**Ren18, JAC+20**] and define

$$\varphi(x,y) = \left(\frac{x_Q x^2 - x}{x - x_Q}, \quad \sqrt{x_Q} \cdot y \cdot \frac{x_Q x^2 - 2x_Q^2 x + x_Q}{(x-x_Q)^2} \right),$$

and

$$(A',B') = \left(2(1-2x_Q^2), B\right).$$

In both cases, the point $(0,0)$ on $E_{A',B'}$ is in the kernel of the dual isogeny $\widehat{\varphi}$. This ensures that the first formula is only used at the start of a (non-backtracking) chain of isogenies.

2.3.2. 4-isogenies

For efficiency purposes, some implementations may prefer to use 4-isogeny formulas instead. For compatibility, these must be equivalent to the compositions of the 2-isogeny formulas above. Let $Q \in E_{A,B}$ be a point of order 4 and let $\varphi : E_{A,B} \to E_{A',B'}$ be the associated 4-isogeny. We distinguish three cases.

$Q = (1, y_Q)$**:** Then $[2]Q = (0,0)$. Composing the first formula with the second one we get:

$$\varphi(x,y) = \left(\frac{1}{A-2} \frac{(x+1)^2(x^2+Ax+1)}{x(x-1)^2}, \quad \frac{1}{\sqrt[3]{A-2}} \cdot y \cdot \frac{(x+1)(x^4-4x^3-2x^2-4Ax^2-4x+1)}{x^2(x-1)^3} \right)$$

and

$$(A',B') = \left(2\frac{6+A}{2-A}, B \right).$$

$Q = (-1, y_Q)$**:** Then, likewise:

$$\varphi(x,y) = \left(\frac{1}{A+2} \frac{(x-1)^2(x^2+Ax+1)}{x(x+1)^2}, \quad \frac{1}{\sqrt[3]{A+2}} \cdot y \cdot \frac{(x-1)(x^4+4x^3-2x^2+4Ax^2+4x+1)}{x^2(x+1)^3} \right)$$

and

$$(A',B') = \left(2\frac{6-A}{2+A}, B \right).$$

$Q = (x_Q, y_Q)$ **with** $x_Q \neq \pm 1$**:** Then

$$\varphi(x,y) = \left(\frac{(x(x_Q^2+1) - 2x_Q)(xx_Q - 1)^2 x}{(2xx_Q - x_Q^2 - 1)(x-x_Q)^2}, \right.$$
$$\left. x_Q^2 \sqrt{2(x_Q^3 + x_Q)} \cdot y \cdot \frac{(xx_Q-1)\big(8x^2x_Q^2 + (x_Q^2+1)(x^4 - 4x^3x_Q + 2x^2x_Q^2 - 4xx_Q + 1)\big)}{(x-x_Q)^3(2xx_Q - x_Q^2 - 1)^2} \right)$$

and

$$(A',B') = (2 - 4x_Q^4, B).$$

Note that these formulas differ in a sign from those made popular in [**CLN16, JAC+20**] for SIKE.

2.3.3. Other odd-degree isogenies using Vélu's formulas

We use the adaption of Vélu's formulas [**Vél71**] to Montgomery curves from [**CH17, MR18**] to compute odd prime degree isogenies with complexity $O(\ell)$. We follow the exposition of [**BDLS20**].

Let ℓ be an odd prime and $Q \in E_{A,B}$ a point of order ℓ. We write $x_{[i]Q}$ for the x-coordinate of $[i]Q$. For computing the unique (up to isomorphisms) isogeny $\varphi : E_{A,B} \to E_{A',B'}$ with kernel $\ker(\varphi) = \langle Q \rangle$, we define the polynomial

$$h_S(x) = \prod_{s \in S} (x - x_{[s]Q})$$

with $S = \{1, 2, \ldots, (\ell-1)/2\}$. Then we can compute the codomain curve by $A' = 2(1+d)/(1-d)$ with

$$d = \left(\frac{A-2}{A+2}\right)^{\ell} \cdot \left(\frac{h_S(1)}{h_S(-1)}\right)^{8}, \quad \text{and} \quad B' = B.$$

Given a point $P \notin \ker(\varphi)$ with $P = (x_P, y_P)$, we can compute its image as

$$P' = \varphi(x_P, y_P) = (f(x_P), \beta y_P f'(x_P)),$$

where $\beta = h_s(0) \cdot \prod_{i=1}^{(\ell-1)/2} x_{[i]Q}$, $f'(x)$ is the derivative of $f(x)$, and

$$f(x) = \frac{x^{\ell} \cdot h_S(1/x)^2}{h_S(x)^2}.$$

In particular, we compute all necessary kernel points $Q, [2]Q, \ldots, [(\ell-1)/2]Q$ and evaluate the formulas above at a total cost of $O(\ell)$

2.3.4. Odd-degree isogenies using $\sqrt{\text{élu}}$

Following [**BDLS20**], we can compute and evaluate ℓ-isogenies in asymptotic complexity $\widetilde{O}(\sqrt{\ell})$ through an asymptotically faster evaluation of $h_S(x)$. In practice, the threshold for which $\sqrt{\text{élu}}$ outperforms the formulas from Section 2.3.3 lies around $\ell \approx 100$.

The evaluation of $h_S(x)$ follows a baby step-giant step approach. We deviate from the choice of S from Section 2.3.3 and set $S = \{1, 3, \ldots, \ell-2\}$. For our purposes and input points of order ℓ, these choices are equivalent. We define an index system (I, J) for S using $m = \lfloor \sqrt{\ell - 1}/2 \rfloor$, and $m' = 0$ if $m = 0$ resp. $m' = \lfloor (\ell+1)/4m \rfloor$. We set

$$I = \{2m(2i+1) \mid 0 \leq i \leq m'\} \quad \text{and} \quad J = \{2j+1 \mid 0 \leq j < m\}.$$

With $I \pm J := (I+J) \cup (I-J)$, we then have $S \backslash (I \pm J) = K$ with $K = \{4mm'+1, \ldots \ell-3, \ell-1\}$. If $m > 0$, we have $\#I = m' \leq m+2$, $\#J = m$, and $\#K \leq 2m-1$, hence all cardinalities are in $O(\sqrt{\ell})$.

Furthermore we define the following biquadratic polynomials:

$$\begin{aligned} F_0(x_1, x_2) &= (x_1 - x_2)^2, \\ F_1(x_1, x_2) &= 2\left((x_1 x_2 + 1)(x_1 + x_2) + 2a x_1 x_2\right), \\ F_2(x_1, x_2) &= (x_1 x_2 - 1)^2. \end{aligned}$$

Using these definition, in $\sqrt{\text{élu}}$ the evaluation of h_S for an input α proceeds in the following steps:

(1) Compute the polynomial $h_I(x) = \prod_{i \in I}(x - x_{[i]Q})$.
(2) Compute the polynomial $D_J(x) = \prod_{j \in J} F_0(x, x_{[j]Q})$.
(3) Compute the resultant $\Delta_{I,J} = \text{Res}_x(h_I, D_J) \in \mathbb{F}_q$.
(4) Compute the polynomial $E_J(x) = \prod_{j \in J} \left(F_0(x, x_{[j]Q})\alpha^2 + F_1(x, x_{[j]Q})\alpha + F_2(x, x_{[j]Q})\right)$.
(5) Compute the resultant $R = \text{Res}_x(h_I, E_J) \in \mathbb{F}_q$.
(6) Compute $h_K = \prod_{i \in K}(\alpha - x_{[i]Q}) \in \mathbb{F}_q$.
(7) Output $h_S(\alpha) = h_K \cdot R / \Delta_{I,J}$.

When using efficient polynomial arithmetic, computing polynomials through product trees, and resultants through multi-point evaluation via remainder trees, all of these steps can be computed in complexity $\widetilde{O}(\sqrt{\ell})$. For details we refer to [**BDLS20**].

2.4. Quaternions and ideals

Quaternion algebras are generalizations of Hamilton's quaternions. These are 4-dimensional vector spaces generated by four elements $\{1, i, j, k\}$, and non-commutative algebras with the following multiplication rules:

$$i^2 = a, \qquad j^2 = b, \qquad ij = -ji = k,$$

where a and b are elements of the base field. The case of interest in SQIsign is that of quaternion algebras over the rational numbers $\mathbb{Q}$ ramified at a prime p and ∞. These algebras being all isomorphic, we speak of *the* quaternion algebra ramified at p and ∞, denoted by $B_{p,\infty}$. The prime p in SQIsign is precisely the same as the characteristic of the finite field discussed above. Because $p \equiv 3 \pmod 4$, the basis of $B_{p,\infty}$ can be taken such that

$$i^2 = -1, \qquad j^2 = -p, \qquad ij = -ji = k.$$

Then, elements of $B_{p,\infty}$ are represented as 5-tuples of integers $(a, b, c, d, r) \in \mathbb{Z}^5$, representing

$$\frac{a + bi + cj + dk}{r},$$

where a canonical representation is obtained by reducing the common denominator.

2.4.1. Big integers

SQIsign needs to represent big integers of variable size. The maximum size reached by the integers depends on the system parameters, however it is difficult to estimate, especially for intermediate results. For this reason, a dynamic multi-precision integer library such as GMP[3] is recommended. Future versions of this specification may determine exact bounds on the largest representable integer and thus enable the used of fix-precision big integers.

The operations SQIsign needs to perform on big integers are part of most big integer libraries, and we will thus list them without details:

- Basic arithmetic (addition, multiplication, ...) of integer and rational numbers;
- Uniform sampling of integers from an interval;
- Approximate and exact integer square roots;
- Pseudo-primality testing using the Miller–Rabin test;
- Extended greatest common divisor: given (a, b), find integers (u, v) such that $ua + bv = \gcd(a, b)$;
- Arithmetic modulo integers;
- Chinese remainder theorem (CRT): given integers $(a_1, \ldots, a_n)$ and coprime moduli $(m_1, \ldots, m_n)$, compute $0 \leq a < m_1 \cdots m_n$ such that $a = a_i \bmod m_i$;
- Legendre, Jacobi and Kronecker symbols;
- Square roots modulo primes.

With the exception of CRT and modular square roots, all these algorithms are implemented in GMP, which is the big integer library used by SQIsign's reference implementation.

2.4.2. Basic integer linear algebra

SQIsign needs to manipulate several integer matrices of small dimension. The most common operations are on 2×2 and 4×4 matrices, with occasional computations on $4 \times n$ or $n \times 4$ matrices for larger n (up to $n = 16$).

Depending on the use case, these matrices may be seen as having coefficients in $\mathbb{Z}$, in $\mathbb{Z}/q^e\mathbb{Z}$ for some prime q and some exponent e, or in $\mathbb{Z}/d\mathbb{Z}$ for some integer d with unknown factorization.

[3] https://gmplib.org/

2.4.2.1. Basic operations on integer vectors and matrices. Basic operations such as matrix-vector and matrix-matrix multiplication can be implemented using the schoolbook method.

For determinants and inversion of 2×2 matrices the standard formulas can be used:

$$\det \begin{pmatrix} a & b \\ c & d \end{pmatrix} = ad - bc, \qquad \begin{pmatrix} a & b \\ c & d \end{pmatrix}^{-1} = \frac{1}{ad-bc} \begin{pmatrix} d & -b \\ -c & a \end{pmatrix}.$$

For determinants and inversion of 4×4 matrices a similar Laplacian formula [**Ebe08**] computes the determinant and the adjugate matrix using 78 ring operations.

2.4.2.2. Hermite normal form. The Hermite normal form (HNF) is a generalization of the reduced echelon form for matrices with integer coefficients. A matrix H is said to be in (column-style) HNF if it satisfies the following conditions:

- It is upper triangular (i.e., $h_{ij} = 0$ for $j < i$) and any columns of zeros are located to the left;
- The leading coefficient, or *pivot*, of a nonzero column is always strictly below of the leading coefficient of the column before it; moreover, it is positive.
- The elements to the left of pivots are zero and elements to the right of pivots are nonnegative and strictly smaller than the pivot.

A matrix A is said to have H for HNF if H is in HNF and there exists a unimodular matrix U such that $AU = H$. Then, A and H have the same column space, and H is unique, thus giving a canonical representation for A's column space. An algorithm for computing the HNF of arbitrary matrices is presented in [**Coh93**, § 2.4.2]. In SQIsign this algorithm is used on 4×4 and 4×8 integer matrices.

2.4.2.3. Kernels of modular integer matrices. SQIsign needs to compute right kernels of matrices with coefficients in $\mathbb{Z}/2^e\mathbb{Z}$, $\mathbb{Z}/q\mathbb{Z}$ (prime q) and $\mathbb{Z}/d\mathbb{Z}$, i.e., given a matrix A, find a matrix K such that $AK = 0$.

For the case of $\mathbb{Z}/q\mathbb{Z}$ with prime q, computing a echelon form (see [**Coh93**, § 2.3.1]) is sufficient. However a more general algorithm is needed for matrices modulo an arbitrary integer d: the *Howell normal form* is a generalization of the HNF for matrices over $\mathbb{Z}/d\mathbb{Z}$. Like the HNF, it preserves the column space and is canonical. A simple algorithm to compute the Howell form is described in [**SM98**, § 3]: given a matrix A it computes its Howell form H and a matrix U such that $AU = H$. Unlike the HNF, H may have more columns than A. Once H is computed, it is easy to compute its kernel K', and then $K = UK'$ is the kernel of A.

2.4.2.4. LLL. Finally, SQIsign needs to compute Lenstra–Lenstra–Lovász (LLL) reduced forms of 4×4 integer matrices. Given a matrix B with columns $\mathbf{b}_1, \ldots, \mathbf{b}_n$, define its Gram-Schmidt vectors as

$$\mathbf{b}_i^\star = \mathbf{b}_i - \sum_{j=1}^{i-1} \mu_{i,j} \mathbf{b}_j^\star,$$

where

$$\mu_{i,j} = \frac{\langle \mathbf{b}_i, \mathbf{b}_j^\star \rangle}{\langle \mathbf{b}_j^\star, \mathbf{b}_j^\star \rangle}$$

and $\langle \cdot, \cdot \rangle$ denotes a norm (for example the usual Euclidean norm, or the reduced norm of $B_{p,\infty}$). Then B is said to be LLL-reduced if

- $|\mu_{i,j}| < 1/2$ for $1 \leq j < i \leq n$, and
- $|\mathbf{b}_i^\star| \geq (3/4 - \mu_{i,i-1}^2)|\mathbf{b}_{i,j}^\star|^2$.

In SQIsign, LLL bases are used to produce short and/or close vectors. The LLL algorithm takes as input an arbitrary matrix and produces an LLL-reduced one using column operations. See, for example [**Coh93**, § 2.6.1].

2.4.3. Close vector enumeration in 2-dimensional lattices

The "strong approximation" algorithm 13 described in Section 2.5.1.4 requires enumerating vectors in a 2-dimensional integral lattice L close to a given target vector t, until one satisfying a certain condition is found. The distance is expressed in this case by a norm N_q defined as $x^2 + qy^2$ with q a given small positive integer.

SQIsign first finds a short basis of L using Gauss' method as described in [**Coh93**, § 1.3.4]. Then it finds a vector v_0 close to the target using Babai's nearest plane algorithm [**Bab86**]. Finally it enumerates short vectors $v \in L$ using the Fincke-Pohst algorithm [**FP85**] (see also [**Coh93**, § 2.7.3]), and outputting $v_0 + v$.

We now give more details on each of the steps.

2.4.3.1. Finding a short basis. Algorithm 1.3.14 in [**Coh93**, § 1.3.4] finds a single short vector. It is easily adapted to output a short basis for the norm N_q, as described in Algorithm 5. We denote by $\lfloor x \rceil$ the rounding of x to the nearest integer. When $u = (u_0, u_1)$ is a 2-vector, $N_q(u)$ is the value of the norm $u_0^2 + qu_1^2$, and we denote by B_q the associated bilinear form.

Algorithm 5 ShortBasis(b_0, b_1)

Input: b_0, b_1 an integral lattice basis, q positive integer
Output: β_1, β_0 short basis of the same lattice for norm N_q, with $N_q(\beta_1) \leq N_q(\beta_0)$

$\beta_0, \beta_1 := b_0, b_1$
if $N_q(\beta_0) < N_q(\beta_1)$ **then**
  $\beta_0, \beta_1 := \beta_1, \beta_0$
end if
while True **do**
  $r := \left\lfloor \frac{B_q(\beta_0, \beta_1)}{N_q(\beta_1)} \right\rceil$
  $\gamma := \beta_0 - r\beta_1$
  if $N_q(\gamma) < N_q(\beta_1)$ **then**
    $\beta_0, \beta_1 := \beta_1, \gamma$
  else
    Break
  end if
end while
if $N_q(\gamma) < N_q(\beta_0)$ **then**
  $\beta_0 := \gamma$
end if
return (β_1, β_0)

2.4.3.2. Finding the closest vector. After a reduced basis β_1, β_0 of L has been computed, we find vectors in L close to a target vector t using Babai's nearest plane algorithm [**Bab86**], which in dimension 2 consists of only 2 projections, as described in Algorithm 6.

Algorithm 6 ClosestVector(β_1, β_0, q, t)

Input: q a positive integer, β_1, β_0 a reduced integral lattice basis such that $N_q(\beta_1) < N_q(\beta_0)$, t a vector with integer coefficients
Output: c vector in lattice generated by β_1, β_0 close to t for N_q

$\mu_1 := N_q(\beta_1)\beta_0 - B_q(\beta_0, \beta_1)\beta_1$
$c := t - \left\lfloor \frac{B_q(\mu_1, t) N_q(\beta_1)}{N_q(\mu_1)} \right\rceil \beta_0$
$c := c - \left\lfloor \frac{B_q(\beta_1, c)}{N_q(\beta_1)} \right\rceil \beta_1$
return c

2.4.3.3. Enumerating close vectors. The last step is the enumeration of short vectors in the lattice for the norm N_q. For this SQIsign uses the Fincke-Pohst (see algorithm 2.7.5 in [**Coh93**, § 1.3.4]). Our variant, described in Algorithm 7, takes as input a lattice basis for L, a target vector t, and a vector c close to t, and enumerates short vectors $v \in L$ to find close vector $c + v$ to t. Other arguments are a bound B on the largest distance from t, and a bound m on the maximum number of vectors returned. We denote by $\sqrt{x}$ the lower integer part of the positive square root of x. We use the Python-generators-inspired **yield** keyword to indicate that the algorithm outputs a vector and then continues the enumeration.

Algorithm 7 EnumerateCloseVectors(L, q, t, c, m, B)

Input: q positive integer, $L = (b_0, b_1)$ a lattice, t target vector, $c \in L$ a vector close to t, m maximal number of tries, B norm bound
Output: A list of vectors $v \in L$ close to t

$i := 0$
$d := t - c$
$a, b, c := N_q(b_0), 2B_q(b_0, b_1), N_q(b_1)$
if $4ac - b^2 \leq 0$ **then**
  Abort
end if
$B_e := B$
if $B - N_q(d) > 0$ **then**
  $e := B - N_q(d)$
end if
$B_y := \left\lfloor \frac{\sqrt{4a^2 B_e} + 1}{\sqrt{4a^a c - b^2}} \right\rfloor + 1$
$y := -B_y - 1$
while $(y < B_y)$ and $(i < m)$ **do**
  $y := y + 1$
  $B_x := \left\lfloor \frac{2a(1+\sqrt{4a^2 B_e + 4ca^2 y^2 - b^2 y^2}) - by\sqrt{4a^3}}{2a\sqrt{4a^3}} \right\rfloor + 1$
  $x := -\left\lfloor \frac{2a(1+\sqrt{4a^2 B_e + 4ca^2 y^2 - b^2 y^2}) + by\sqrt{4a^3}}{2a\sqrt{4a^3}} \right\rfloor - 2$
  while $(x < B_x)$ and $(i < m)$ **do**
    $x := x + 1$
    $i := i + 1$
    if $N_q(d - xb_0 - yb_1) \leq B$ **then**
      yield $c + xb_0 + yb_1$
    end if
  end while
end while

2.4.4. Quaternions and lattices

SQIsign keeps track of elements $\alpha \in B_{p,\infty}$ and of full-rank lattices $\alpha_1 \mathbb{Z} + \alpha_2 \mathbb{Z} + \alpha_3 \mathbb{Z} + \alpha_4 \mathbb{Z}$ where $(\alpha_1, \ldots, \alpha_4)$ is a vector-space basis of $B_{p,\infty}$. We now describe how these objects are represented and manipulated.

2.4.4.1. Basic quaternion arithmetic. We already mentioned quaternions are represented by 5 integers as

$$\alpha = \frac{a + bi + cj + dk}{r}.$$

The basic arithmetic operations (addition, multiplication, etc.) are handled according to the usual rules, reducing common denominators where necessary. Multiplication follows from the three axioms $i^2 = -1$, $j^2 = -p$, $ij = -ij = k$; we report the multiplication table in Figure 2.4.4.1.

	1	i	j	ij
1	1	i	j	ij
i	i	-1	ij	$-j$
j	j	$-ij$	$-p$	pi
ij	ij	j	$-pi$	$-p$

FIGURE 1. Multiplication table for $B_{p,\infty}$, the quaternion algebra ramified at a prime p and $+\infty$.

Some other specific operations are as follows. The conjugate $\bar{\alpha}$ of α is the element

$$\bar{\alpha} = \frac{a - bi - cj - dk}{r}.$$

Their reduced trace is

$$\mathrm{tr}(\alpha) = \mathrm{tr}(\bar{\alpha}) = \alpha + \bar{\alpha} = \frac{2a}{r},$$

and their reduced norm is

$$\mathrm{nrd}(\alpha) = \mathrm{nrd}(\bar{\alpha}) = \alpha\bar{\alpha} = \frac{a^2 + b^2 + p(c^2 + d^2)}{r^2}.$$

2.4.4.2. Lattices. A lattice is defined by a basis $(\alpha_1, \alpha_2, \alpha_3, \alpha_4)$ of $\mathbb{Q}$-linearly independent quaternions. By convention throughout this specification elements are represented as *columns* of a matrix L, so that

$$(\alpha_1\ \alpha_2\ \alpha_3\ \alpha_4) = (1\ i\ j\ k) \cdot L$$

is the row vector describing the basis.[4]

The dual of a lattice L is defined as

$$L^\star = \{f \in (\mathrm{span}(L))^\star \mid \forall x \in L, f(x) \in \mathbb{Z}\}.$$

In practice, because all lattices we consider have full rank, the matrix representing $L^\star$ is the transpose of L^{-1}.

To ensure a canonical representation and simplify operations, the SQISIGN reference implementation puts all lattices in HNF. Given this representation, several operations on lattices are straightforward to compute [**Coh93**, § 2.4.3]:

Equality: Check that L_1 and L_2 have the same HNF.

Union: If L_1 and L_2 are lattices, concatenate their matrices $L_1|L_2$ and compute the HNF to obtain $L_1 + L_2$.

Intersection: If L_1 and L_2 are lattices, compute their dual lattices $L_1^\star$ and $L_2^\star$; then $L_1 \cap L_2$ is the dual of $L_1^\star + L_2^\star$.

Multiplication: If L_1 and L_2 are lattices, their product L_1L_2 is computed, e.g., by writing the right multiplication matrices $A_1, \ldots, A_4$ of the generators $\alpha_1, \ldots, \alpha_4$ of L_2, and then computing the union

$$A_1L_1 + A_2L_1 + A_3L_1 + A_4L_1.$$

Containment: Given an element $\alpha \in B_{p,\infty}$ and a lattice L, checking whether $\alpha \in L$ is done by solving a linear system $LX = \alpha$ and verifying that X has integer entries. Having L in HNF further simplifies the linear solving. The same algorithm can be used to check that a lattice L' is contained in L.

Index: When $L_1 \subset L_2$, the index of L_1 in L_2, denoted by $[L_2 : L_1]$ is the order of the finite quotient group L_2/L_1. This value equals $|\det(L_1)/\det(L_2)|$ and is straightforward to compute given L_1 and L_2 in HNF.

Right transporter (colon lattice): This is the most complex operation SQISIGN needs to perform on lattices. The right transporter, also known as colon lattice, is the lattice T of elements $\alpha \in B_{p,\infty}$ such that $L_1 \cdot \alpha \subset L_2$, which can be understood as an inverse to lattice multiplication [**Voi21**, Chap. 15]. We first compute the lattices

$$A_1 = L_1, \quad A_i = L_1 \cdot i, \quad A_j = L_1 \cdot j, \quad A_k = L_1 \cdot k$$

using the algorithm for lattice multiplication. Then we compute the dual lattice $L_2^\star$ and evaluate $B_x = L_2^\star(A_x)$ for each A_x, which amounts to computing the matrix product $L_2^{-1}A_x$. Then T consists of those elements $a + bi + cj + dk$ such that $aB_1 + bB_i + cB_j + dB_k$ is integral. Because we know that this module is

[4] In practice the reference implementation takes common denominators and represents L as an integer matrix M and a common denominator r, so that $L = M/r$. This can be considered as an implementation detail.

contained in $1/\det(L_1)\cdot\mathbb{Z}^4$, we can find it as the kernel of the mapping $(a,b,c,d) \mapsto aB_1+bB_i+cB_j+dB_k \mod \mathrm{num}(\det(L_1))$, using the algorithms described in Section 2.4.2.3.

2.4.5. Quaternion orders and ideals

An *order* is a lattice of $B_{p,\infty}$ that is also a subring. Elements of an order $\mathcal{O}$ are said to be integral, since they have reduced norm and trace in $\mathbb{Z}$. An order is called *maximal* when it is not contained in any other larger order.

An *integral ideal*, henceforth only called an ideal, is a sublattice of an order. The left order of an ideal is defined as $\mathcal{O}_L(I) = \{\alpha \in \mathcal{B}_{p,\infty} \mid \alpha I \subset I\}$ and similarly for the right order $\mathcal{O}_R(I)$. Then I is said to be a left ideal of $\mathcal{O}_L(I)$. An ideal with left order $\mathcal{O}_L$ and right order $\mathcal{O}_R$ is called a *connecting ideal* of $\mathcal{O}_L$ and $\mathcal{O}_R$. The norm of an ideal I is the greatest common divisor of the norms of its elements; it is an integer equal to $\sqrt{[O_L(I) : I]} = \sqrt{[O_R(I) : I]}$. Any ideal can be written as $I = \mathcal{O}_L(I)\alpha + \mathcal{O}_L(I)\mathrm{nrd}(I)$ for some $\alpha \in \mathcal{O}_L(I)$, and similarly for $\mathcal{O}_R(I)$. We simplify this notation by writing $\mathcal{O}\alpha + \mathcal{O}N = \mathcal{O}\langle\alpha, N\rangle$ for any order $\mathcal{O}$.

The product IJ of ideals I and J satisfying $\mathcal{O}_R(I) = \mathcal{O}_L(J)$ is the ideal generated by the products of pairs in $I \times J$. It follows that IJ is also an (integral) ideal and $\mathcal{O}_L(IJ) = \mathcal{O}_L(I)$ and $\mathcal{O}_R(IJ) = \mathcal{O}_R(J)$. The ideal norm is multiplicative with respect to ideal products.

We define an equivalence on orders by conjugacy and on left $\mathcal{O}$-ideals by right scalar multiplication. Two orders $\mathcal{O}_1$ and $\mathcal{O}_2$ are equivalent if there is an element $\beta \in \mathcal{B}^\star_{p,\infty}$ such that $\beta\mathcal{O}_1 = \mathcal{O}_2\beta$. Two left $\mathcal{O}$-ideals I and J are equivalent if there exists $\beta \in \mathcal{B}^\star_{p,\infty}$, such that $I = J\beta$. If the latter holds, then it follows that $\mathcal{O}_R(I)$ and $\mathcal{O}_R(J)$ are equivalent since $\beta\mathcal{O}_R(I) = \mathcal{O}_R(J)\beta$.

2.4.5.1. Basic operations on ideals. We represent ideals by their lattices and compute equality, membership, union, intersection and multiplication like for them. We can use these elementary operations to compute the ideal $\mathcal{O}\langle\alpha, N\rangle$: use lattice multiplication to compute $\mathcal{O}\alpha$ and $\mathcal{O}N$, then lattice union to compute $\mathcal{O}\langle\alpha, N\rangle$. Inversely, a generator α such that $I = \mathcal{O}\langle\alpha, N\rangle$ can be found by taking arbitrary elements in I until one such that $\gcd(\mathrm{nrd}(\alpha), N^2) = N$ is found.

The left and right order of an ideal: The right (resp. left) order of an ideal I is by definition the right (resp. left) transporter of I into itself.

Computing isomorphism of ideals: Two left $\mathcal{O}$-ideals I, J are isomorphic as $\mathcal{O}$-modules if and only if they are equivalent, i.e. if there exists a $\beta \in \mathcal{B}^\star_{p,\infty}$ such that $I = J\beta$. Given equivalent left $\mathcal{O}$-ideals I and J, finding an element β such that $I = J\beta$ can be done by first computing the right transporter T such that $JT \subset I$, followed by finding a short vector β in T (e.g., using LLL); all vectors in T are multiples have norm a multiplie of $N(I)/N(J)$, and a vector in T of norm exactly $N(I)/N(J)$ exists if and only if I and J are equivalent.

Computing a connecting ideal: Given two orders $\mathcal{O}_L$ and $\mathcal{O}_R$, we compute a connecting ideal as

$$I = (\mathcal{O}_L + \mathcal{O}_L\mathcal{O}_R)N$$

, where $N = \mathrm{nrd}(\mathcal{O}_L \cap \mathcal{O}_R)$.

Pullback and pushforward of ideals: We recall two definitions from [DKL+20, Lemma 3]. Given two ideals I, J, such that $\mathcal{O}_R(J) = \mathcal{O}_L(I)$ and having coprime norm, we define the pullback ideal to be the $\mathcal{O}_L(J)$-ideal

$$[J]^*I = JI + \mathrm{nrd}(I)\mathcal{O}_L(J).$$

Similarily, when I, J are two left $\mathcal{O}$-ideals of coprime norm, we define the pushforward ideal to be the left $\mathcal{O}_R(J)$-ideal

$$[J]_*I = J^{-1}(J \cap I).$$

It is readily verified that $[J]^*([J]_*I) = I$. The motivation for these definitions comes from the correspondence between ideals and isogenies (see Section 2.6), where the corresponding definitions are more natural.

2.4.5.2. Finding equivalent ideals of small norm. Given an integral ideal I, we SQIsign often need to find an equivalent ideal J (i.e., $I = J\alpha$, where $\alpha \in \mathcal{B}^{\star}_{p,\infty}$) with different norm. To do so, we will employ the surjection

$$\chi_I(\alpha) = I\frac{\overline{\alpha}}{\mathrm{nrd}(I)}$$

from $I\backslash\{0\}$ to the set of ideals J equivalent to I.

One example of this, is when SQIsign needs equivalent ideals, whose norm is prime and bounded. This is done with RandomEquivalentPrimeIdeal, which is given in Algorithm 8. The algorithm samples constants c_i and constructs $\beta = \sum_{i=1}^{4} c_i\alpha_i$ where the α_i is a reduced basis of I, before outputting $J = \chi_I(\beta)$ if the norm of J is prime. This process is repeated a maximal number of `KLPT_equiv_num_iter` times.

Size of outputs. Heuristically, we can expect the norm of the output ideal to be $\approx \sqrt{p}$. However, this heuristic fails whenever there exists an equivalent ideal of I (for instance I itself) of norm significantly smaller than $\sqrt{p}$. This has implication in, for instance SpecialEichlerNorm (Algorithm 15), where we do not run RandomEquivalentPrimeIdeal on ideals that have too small norm.

Algorithm 8 RandomEquivalentPrimeIdeal(I)

Input: I a left $\mathcal{O}$-ideal
Output: $J \sim I$ of small prime norm
Output: `found` a boolean indicating whether computation was successful

Initialise `counter` $:= 0$ and `found` $:=$ False
Compute a LLL-reduced basis $\alpha_1, \alpha_2, \alpha_3, \alpha_4$ of I
while not `found` and `counter` $<$ `KLPT_equiv_num_iter` **do**
  `counter` $:=$ `counter` $+ 1$
  Sample c_1, c_2, c_3, c_4 from $[-b, \ldots, b]$, for bound $b :=$ `KLPT_equiv_bound_coeff`
  $\beta := \sum_{i=1}^{4} c_i\alpha_i$
  $J := \chi_I(\beta)$
  if $\mathrm{nrd}(J)$ is prime **then**
    `found` $:=$ True
  end if
end while
return J, `found`

2.5. Solving norm equations

An important part of both the key generation and signing procedure in SQIsign, is the ability to solve norm equations quaternion orders and ideals. Throughout this section, $\mathcal{O}_0$ will refer to a p-extremal maximal order, i.e. a maximal order with a distinguished quadratic subring R, so that

$$R + jR \subseteq \mathcal{O}_p,$$

while $\mathcal{O}$ refers to an arbitrary maximal order. Further, several constants (defined in Chapter 7) will be used as part of the algorithms; these constants are of the form `KLPT_xxx`.

In Section 2.5.2, we will define the three KLPT-based procedures for solving norm equations. Specifically, KeyGenKLPT is used during key generation, and solves a norm equation in a left $\mathcal{O}_0$-ideal, SpecialEichlerNorm solves a norm equation in $\mathcal{O}$, while finally, SigningKLPT solves a norm equation in a left $\mathcal{O}$-ideal. Before that, in Section 2.5.1, we describe the necessary subroutines that make up the three algorithms.

2.5.1. Building blocks

In this section, we describe the necessary building blocks for solving norm equations in quaternion orders and ideals.

2.5.1.1. Cornacchia's algorithm. Cornacchia's algorithm [Cor08] allows us to efficiently solve norm equations of the form $x^2 + ny^2 = m$, provided we know the factorization of m. In SQIsign, the function Cornacchia checks whether m is easy to factor first, before possibly running Cornacchia's algorithm to find x, y (see also [MN90]). The whole algorithm, depicted in Algorithm 9, returns x, y and `found`, a boolean indicating whether or not a solution was found.

Algorithm 9 Cornacchia(m)

Input: $m \in \mathbb{Z}$
Output: x, y such that $x^2 + y^2 = m$
Output: `found` a boolean indicating whether a solution was found
1: $x := 0, y := 0,$ `found` $:=$ False
2: Find the 100-smooth part m' of m using trial division
3: **if** m/m' is prime **then**
4: Run Cornacchia's algorithm to find x, y such that $x^2 + y^2 = m$, and set `found` $:=$ True if a solution was found
5: **end if**
6: **return** $x, y,$ `found`

2.5.1.2. Representing integers by the special extremal order. Cornacchia's algorithm allows us to efficiently solve norm equations of the form $x^2 + ny^2 = m$. This makes it straight forward to solve norm equations in the suborder $R + jR \subseteq \mathcal{O}_0$ (where $R = \mathbb{Z}[i]$ is the distinguished quadratic subring of $\mathcal{O}_0$), whose norm form is given by

$$F(t, x, y, z) = x^2 + y^2 + p(z^2 + w^2).$$

The general idea is to sample z, w, before using Cornacchia to see if we can find x, y such that $x^2+y^2 = T-p(z^2+w^2)$ This algorithm is depicted in Algorithm 10, together with an extension introduced in [DLLW23, §6], which samples solutions in the whole order $\mathcal{O}_0$. As pointed out in [DLLW23], finding solutions in the full order is essential for zero-knowledge in SQIsign (see also Chapter 9).

Algorithm 10 FullRepresentInteger(M)

Input: $M \in \mathbb{Z}$ such that $M > p$
Output: $\gamma \in \mathcal{O}_0$ with $\mathrm{nrd}(\gamma) = M$
Output: `found` a boolean indicating whether a solution was found
1: Initialise `counter` $:= 0$ and `found` $:=$ False
2: **while** not `found` and `counter` $<$ `KLPT_repres_num_gamma_trial` **do**
3: Sample z' from $[-m, \ldots, m]$ for $m := \left\lfloor \sqrt{\frac{4M}{p}} \right\rfloor$
4: Sample w' from $[-m', \ldots, m']$ for $m' := \left\lfloor \sqrt{\frac{4M - z'^2}{p}} \right\rfloor$
5: Set $M' = 4M - p((z')^2 + (w')^2)$
6: $x', y',$ `found` $:=$ Cornacchia(M')
7: **if** `found` $= 0$ or $x' \neq w' \bmod 2$ or $z' \neq y' \bmod 2$ **then**
8: `found` $:=$ False
9: `counter` $:=$ `counter` $+ 1$
10: **end if**
11: **end while**
12: **if** `found` **then**
13: Set k to be the biggest scalar such that $\gamma/k \in \mathcal{O}_0$
14: $\gamma := (x' + iy' + jz' + kw')/k$
15: **else**
16: $\gamma := 0$
17: **end if**
18: **return** $\gamma,$ `found`

2.5.1.3. Reduction to linear systems. The higher level norm equation algorithms always involve solving some linear system of equations. The general algorithm for this step is given in EichlerModConstraint; further, another similar subroutine is FindLinearCombination, used as part of the IdealToIsogenyEichler$_{\ell^\bullet}$ algorithm (see Section 2.6). Both algorithms use linear algebra in $\mathbb{Z}/N\mathbb{Z}$ to find a quaternion of a special form, satisfying some constraint.

In EichlerModConstraint, we consider quaternion elements $\alpha = x+yi+zj+wk$ with $[x,y,z,w] \in \mathbb{Z}^4$ as integer column vectors (note that by clearing out the denominators, all coefficients really lie in $\mathbb{Z}$). In FindLinearCombination, we instead compute a basis $\langle \alpha_1, \alpha_2, \alpha_3, \alpha_4 \rangle = \mathcal{O}$, and compute vectors $[x,y,z,w] \in \mathbb{Z}^4$ corresponding to $\alpha \in \mathcal{O}$, given by $\alpha = x\alpha_1 + y\alpha_2 + z\alpha_3 + w\alpha_4$. In theory, the representation in EichlerModConstraint is just a special case of this for elements in the order $\langle 1, i, j, k \rangle$, however, we make the distinction in these algorithms explicit, as, in SQIsign, quaternions are already represented as a decomposition along the basis $1, i, j, k$.

Note also that in EichlerModConstraint, it may be a necessary constraint for the output $\mu_0 = \gamma j(C+iD)\delta$ to satisfy $\mu_0 \in I$ (which is stronger than $\mu_0 \in \mathbb{Z}+I$). This happens automatically if $\mathrm{nrd}(\gamma)$ is divisible by $\mathrm{nrd}(I)$; hence, in all cases where this constraint is necessary, we run the algorithm with this additional condition in mind. The algorithm also takes as input a bit `divisible`, which indicates if this condition is true.

Algorithm 11 EichlerModConstraint$(I, \gamma, \delta, \texttt{Divisible})$

Input: I a left $\mathcal{O}$-ideal of norm N
Input: $\gamma, \delta \in \mathcal{B}_{p,\infty}$
Input: `Divisible` a boolean indicating whether $N \mid \mathrm{nrd}(\gamma)$
Output: $C, D \in \mathbb{Z}$ such that $\gamma j(C+iD)\delta \in \mathbb{Z}+I$

Let d_1 be the denominator of $\gamma j i \delta$
Let d_2 be the denominator of I
Let d_3 be the denominator of $\gamma j \delta$
Let $\beta_1, \beta_2, \beta_3, \beta_4$ denote a basis of I in Hermite normal form
if `Divisible` **then**
    $\mathbf{M} := [\gamma j \delta d_1 d_2, \gamma j i \delta d_2 d_3, \beta_2 d_1 d_3, \beta_3 d_1 d_3] \in \mathbb{M}_{4\times 4}(\mathbb{Z}/N\mathbb{Z})$
    Compute $[C, D, \_, \_]^{\mathrm{T}} \in \ker \mathbf{M}$
else
    $\mathbf{M} := [\gamma j \delta d_1 d_2, \gamma j i \delta d_2 d_3, d_1 d_2 d_3, \beta_2 d_1 d_3, \beta_3 d_1 d_3] \in \mathbb{M}_{4\times 5}(\mathbb{Z}/N\mathbb{Z})$
    Compute $[C, D, \_, \_, \_]^{\mathrm{T}} \in \ker \mathbf{M}$
end if
return C, D

Algorithm 12 FindLinearCombination(β, I, K)

Input: $\beta \in \mathcal{O}$
Input: I, K left $\mathcal{O}$ ideals of norm ℓ^f
Output: $C, D \in \mathbb{Z}$ such that $\alpha(C+D\beta) \in K$ and $\gcd(C, D, \ell) = 1$, where $I = \mathcal{O}\langle \alpha, \ell^f \rangle$

Compute α such that $I = \mathcal{O}\langle \alpha, \ell^f \rangle$
Let $\beta_1, \beta_2, \beta_3, \beta_4$ denote a basis of I in Hermite normal form
Let $\mathbf{A} = [\alpha_1, \alpha_2, \alpha_3, \alpha_4]$ denote a basis of $\mathcal{O}$
Compute $\mathbf{v}_\alpha \in \mathbb{Z}^4$ such that $\mathbf{A}\mathbf{v}_\alpha = \alpha$
Compute $\mathbf{v}_{\alpha\beta} \in \mathbb{Z}^4$ such that $\mathbf{A}\mathbf{v}_{\alpha\beta} = \alpha\beta$
Compute $\mathbf{v}_{\beta_2} \in \mathbb{Z}^4$ such that $\mathbf{A}\mathbf{v}_{\beta_2} = \beta_2$
Compute $\mathbf{v}_{\beta_3} \in \mathbb{Z}^4$ such that $\mathbf{A}\mathbf{v}_{\beta_3} = \beta_3$
$\mathbf{M} := [\mathbf{v}_\alpha, \mathbf{v}_{\alpha\beta}, \mathbf{v}_{\beta_2}, \mathbf{v}_{\beta_3}] \in \mathbb{M}_{4\times 4}(\mathbb{Z}/N\mathbb{Z})$
Compute $[C, D, \_, \_]^{\mathrm{T}} \in \ker \mathbf{M}$
return C, D

2.5.1.4. StrongApproximation. The final step of every high level norm equation algorithm is to go from an element $\mu_0 := j(C + iD)$ satisfying $\gamma\mu_0\delta \in L$ where L is the (multiplicative) lattice we want to solve the norm equation in, to an element $\mu := \lambda\mu_0 + N\mu_1$, where $\lambda \in \mathbb{Z}, \mu_1 \in \mathcal{O}_0$, which automatically satisfies the same constraint and also solves the norm equation. The original idea is to look for $\mu_1 := x_1 + y_1\omega + j(z_1 + w_1\omega) \in R + jR$, solving for the particularly simple norm form

$$\mathrm{nrd}(\mu) := N^2 f(x_1, y_1) + pf(C + Nz_1, D + Nw_1),$$

by first solving it modulo N. Algorithm 13 shows how to do this, including an improvement by [**PS18**] for finding shorter solutions, and by [**DLLW23**] for finding μ_1 in the whole order.

Algorithm 13 $\mathsf{FullStrongApproximation}_{N_\mu}(\mathcal{O}_0, N, C, D, \lambda, \texttt{max\_tries}, K_{\text{OPT}})$

Input: $\mathcal{O}_0$ a p-extremal maximal order with distinguished suborder $\mathbb{Z}[\sqrt{-q}] + j\,\mathbb{Z}[\sqrt{-q}]$
Input: N a prime
Input: $C, D \in \mathbb{Z}$
Input: λ satisfying $\lambda^2 = N_\mu/p(C^2 + qD^2) \pmod N$
Input: `max_tries` a number bounding the number of tries
Input: K_{OPT} an optional left $\mathcal{O}$-ideal
Output: $\mu = \lambda j(C + Di) + N\mu_1$ with $\mu_1 \in \mathcal{O}_0$, such that $\mathrm{nrd}(\mu) = N_\mu$. Additionally, if K_{OPT} is given, then the output also satisfies $\mu \notin \mathbb{Z} + K_{\text{OPT}}$
Output: A boolean indicating whether a solution was found

$c_0 := 2p\lambda C$
$c_1 := (2p\lambda Dq)^{-1} \pmod N$
$c_2 := (T' - \lambda^2 n_{\mu_0})/N$
$c_3 := -c_0 c_1 \pmod N$
$\mathbf{L} := \begin{pmatrix} N & c_3 N \\ 0 & N^2 \end{pmatrix} \in \mathbb{M}_{2\times 2}(\mathbb{Z})$
$\mathbf{v} := [\lambda C, \lambda D + Nc_1c_2]$
$\mathbf{v}' := \mathsf{ClosestVector}(\mathbf{L}, q, \mathbf{v})$
$B := \min(\lceil \log_2(N_\mu/p) \rceil, 3\lceil \log_2(N) \rceil + 10)$
for μ in $\mathsf{EnumerateCloseVectors}(\mathbf{L}, q, \mathbf{v}, \mathbf{v}', \texttt{max\_tries}, B)$ **do**
  if $\mu/2$ is primitive in $\mathcal{O}_0$ and $\mu \notin \mathbb{Z} + K_{\text{OPT}}$ if K_{OPT} is given **then**
    return $\mu/2$, TRUE
  end if
end for
return _, FALSE

2.5.2. The KLPT algorithm and generalisations

Given a $\mathcal{O}_0$-ideal I with $\mathrm{nrd}(I) = N$, the goal of the KLPT algorithm is to find an equivalent ideal J of norm dividing T. This is done by finding an element $\alpha \in I$ of $\mathrm{nrd}(\alpha) \mid T$, and then setting $J := \chi_I(\alpha)$, in other words, by solving a norm equation in I. This is done as follows:

(1) Find an element $\gamma \in \mathcal{O}_0$ having norm dividing NT (using $\mathsf{FullRepresentInteger}$).
(2) Find $C, D \in \mathbb{Z}$ such that $\mu_0 = j(C + iD)$ satisfies $\gamma\mu_0 \in I$ (using $\mathsf{EichlerModConstraint}$).
(3) Compute $\mu = \lambda\mu_0 + N\mu_1$ of norm dividing $NT/\mathrm{nrd}(\gamma)$ (using $\mathsf{FullStrongApproximation}$).
(4) Output $\gamma\mu$.

In SQISIGN, the original KLPT algorithm is used as part of the key generation procedure, here denoted as $\mathsf{KeyGenKLPT}$. See Algorithm 14 for a detailed description. This algorithm follows a similar structure to the standard KLPT algorithm specialised to our application of key generation.

Algorithm 14 KeyGenKLPT(I)

Input: I a left $\mathcal{O}_0$-ideal of small prime norm (bitsize is `KLPT_secret_key_prime_size` $\approx \log(p)/4$)
Output: Generator β of equivalent ideal $J \sim I$ of norm $2^{\texttt{KLPT\_keygen\_length}}$
Output: `found` a boolean indicating whether a solution was found

$N_I := \mathrm{nrd}(I)$
Initialise `counter` := 0 and `found` := False
$k := \lceil \log_2(p) \rceil - \lfloor \log_2(N_I) \rfloor +$ `KLPT_gamma_exponent_center_shift`
$N_\gamma := 2^k$
$N_\mu := 2^{\texttt{KLPT\_keygen\_length}-k}$
while not `found` and `counter` < `KLPT_keygen_num_gamma_trial` **do**
  `counter` := `counter` + 1
  γ, `found` := FullRepresentInteger$(N_I N_\gamma)$
  if not `found` **then**
    Continue
  end if
  $C, D :=$ EichlerModConstraint$(I, \gamma, 1,$ True$)$
  Compute $\alpha := j(C + iD)$
  if $N_\mu/\mathrm{nrd}(\alpha)$ is not a square $\bmod N_I$ **then**
    Continue
  end if
  Set λ to be the square root of $4N_\mu/\mathrm{nrd}(\alpha)$
  `max` := `KLPT_signing_number_strong_approx`
  μ, `found` := FullStrongApproximation$_{4N_\mu}(\mathcal{O}_0, N_I, C, D, \lambda,$ `max`$)$
  if not `found` **then**
    Continue
  end if
end while
return $\gamma\mu$, `found`

Repeating a bounded number of times. The `KeyGenKLPT` algorithm loops through (with new values of γ) a maximal number of `KLPT_keygen_num_gamma_trial` times. If at this point, the algorithm still has not managed to find a valid output, the algorithm fails. In this case, the key generation procedure must be restarted. For details on how this constant is chosen, see Chapter 7.

2.5.2.1. Solving norms using Eichler orders. While the key generation procedure requires solving norm equations in $\mathcal{O}_0$-ideals, hence requiring the original KLPT algorithm, we now describe a simpler algorithm, which solves norm equations in any maximal order $\mathcal{O}$. The idea is to solve the norm equation in the Eichler order $\mathcal{O}_0 \cap \mathcal{O}$. Writing I for the connecting $(\mathcal{O}_0, \mathcal{O})$-ideal, we know that $\mathcal{O}_0 \cap \mathcal{O} = \mathbb{Z} + I$ [**DKL+20**, Proposition 1], allowing us to solve the norm equation in $\mathbb{Z} + I$, which is a bigger lattice than what KLPT solves for. The result of this, is that the output (i.e. the minimal sized norm which we can solve for) of this procedure is smaller than the output of the original KLPT algorithm.

Note also, that we can easily replace $\mathcal{O}_0$ by other p-extremal maximal orders $\mathcal{O}_0'$, in case of failure. SQIsign uses several such p-extremal maximal orders that are stored in a list `extremal_order_list`. If the algorithms is not able to find a solution for any of these alternate orders, the algorithm (which is denoted SpecialEichlerNorm, see Algorithm 15) fails. Note that while this is possible, the parameters can be tuned to make this even extremely rare.

Solving for an isomorphic order. Writing I for the connecting $(\mathcal{O}_0, \mathcal{O})$-ideal, we will in practice find some equivalent ideal $J = I\alpha$ of small, odd norm (unless I already has small, odd norm, in which case we simply try $J = I$, see Subsection 2.4.5.2). This induces an isomorphism $\mathcal{O}_R(J) = \alpha\mathcal{O}\alpha^{-1}$, which allows us to solve the norm equation in $\mathcal{O}_R(J) \cap \mathcal{O} = \mathbb{Z} + J$, before transporting the solution back to $\mathcal{O}$ using the isomorphism between $\mathcal{O}$ and $\mathcal{O}_R(J)$.

Constraint on output. We require our output β to be in $\mathcal{O}\backslash(\mathbb{Z}+K)$. The reason for this constraint has to do with where SQIsign applies Algorithm 15; in Section 2.6, we show how it is used a subroutine of $\mathsf{IdealToIsogenyEichler}_{\ell^\bullet}$ (Algorithm 19). The condition that $\beta \in \mathcal{O}\backslash(\mathbb{Z}+K)$ ensures that the ideal-to-isogeny algorithm is correct and terminates [**DLLW23**, Lemma 8].

Algorithm 15 $\mathsf{SpecialEichlerNorm}_T(\mathcal{O}, I, K)$

Input: I a $(\mathcal{O}_0, \mathcal{O})$-ideal
Input: K a left $(\mathcal{O})$-ideal of norm ℓ
Output: $\beta \in \mathcal{O}\backslash(\mathbb{Z}+K)$ of norm dividing T^2
Output: `found` a boolean indicating that a solution was found

if $\mathrm{nrd}(I)$ odd and $\mathrm{nrd}(I) <$ `KLPT_eichler_smallnorm_bitsize` **then**
  $J := I$
  μ, `found` $:= \mathsf{SpecialEichlerNormFixed}_T(\mathcal{O}_0, J, K)$
end if
if $\mathrm{nrd}(I) >$ `KLPT_eichler_smallnorm_bitsize` **then**
  `counter` $:= 0$
  while not `found` and `counter` $<$ `KLPT_eichler_num_equiv_ideal` **do**
    $J := \mathsf{RandomEquivalentPrimeIdeal}(I)$
    μ, `found` $:= \mathsf{SpecialEichlerNormFixed}_T(\mathcal{O}_0, J, K)$
    `counter` $:=$ `counter` $+ 1$
  end while
end if
if not `found` **then**
  for $\mathcal{O}'_0 \in$ `extremal_order_list` **do**
    $I := I(\mathcal{O}'_0, \mathcal{O})$
    `counter` $:= 0$
    while not `found` and `counter` $<$ `KLPT_eichler_num_equiv_ideal` **do**
      $J := \mathsf{RandomEquivalentPrimeIdeal}(I)$
      μ, `found` $:= \mathsf{SpecialEichlerNormFixed}_T(\mathcal{O}'_0, J, K)$
      `counter` $:=$ `counter` $+ 1$
    end while
  end for
end if
Compute $\alpha \in J$ so that $I = J\alpha$
return $\alpha\mu\alpha^{-1}$, `found`

2.5.2.2. Generalised KLPT for signing. Finally, the signing procedure of SQIsign requires solving norm equations in $\mathcal{O}$-ideals. This is done by the generalised KLPT algorithm, introduced in [**DKL+20**]. Writing I_τ for the connecting $(\mathcal{O}_0, \mathcal{O})$-ideal, and $\mathfrak{O} = \mathcal{O}_0 \cap \mathcal{O}$ this algorithm is based on the fact that when $\beta \in I \cap \mathfrak{O}$, we have the equality

$$[I_\tau]^*(I\beta/N) = ([I_\tau]^* I)\beta/N.$$

In particular, this means that $\beta \in [I_\tau]^* I \cap \mathfrak{O}$. Since $[I_\tau]^* I$ is an $\mathcal{O}_0$-ideal, solving for β in this lattice can be solved by the original KLPT-algorithm, while solving for β in $\mathfrak{O}$ can be solved by the discussion in the previous section. Roughly speaking, SigningKLPT combines the constraint solutions via the Chinese remainder theorem, before using FullStrongApproximation to solve the norm equation. We denote by $\mathsf{CRT}_{M_0,M_1}(x_0, x_1)$ the function which outputs x satisfying $x \equiv x_0 \pmod{M_0}$ and $x \equiv x_1 \pmod{M_1}$, using the Chinese remainder theorem.

The algorithm SigningKLPT (see Algorithm 17) is based on this generalised KLPT algorithm, specialised for the target application of signing.

Algorithm 16 $\mathsf{SpecialEichlerNormFixed}_T(\mathcal{O}_0, J, K)$

Input: $\mathcal{O}_0$ a special p-extremal maximal order
Input: J a $(\mathcal{O}_0, \mathcal{O})$-ideal of norm N_J
Input: K a left $\mathcal{O}$-ideal of norm ℓ
Output: $\beta \in \mathcal{O} \backslash (\mathbb{Z} + K)$ of norm dividing T^2
Output: `found` a boolean indicating that a solution was found

$C, D := \mathsf{EichlerModConstraint}(J, 1, 1, \textsc{False})$
Set $\mathtt{list}_{\mathtt{N}_\beta}$ to be a list of size `KLPT_eichler_number_mu_norm`, containing integers $N_\beta \mid T^2$ such that $N_\beta/(p(C^2 + qD^2))$ is a square mod N_J, and satisfying $N_\beta > pN^3 2^{\texttt{KLPT\_eichler\_strong\_approx\_log\_margin}}$
Initialise `counter` $:= 0$ and `found` $:=$ FALSE
while not `found` and `counter` $<$ `KLPT_eichler_number_mu_norm` **do**
  $N_\beta := \mathtt{list}_{\mathtt{N}_\beta}[\mathtt{counter}]$
  Set λ to be a square root of $4N_\beta/(p(C^2 + qD^2)) \pmod{N_J}$
  `max` $:=$ `KLPT_eichler_number_strong_approx`
  $\beta, \mathtt{found} := \mathsf{FullStrongApproximation}_{4N_\beta}(\mathcal{O}_0, N_J, C, D, \lambda, \mathtt{max}, K)$
  `counter` $:=$ `counter` $+ 1$
end while
return $\beta, \mathtt{found}$

2.6. Converting between ideals and isogenies

In this section, we describe the algorithm for converting between ideals and isogenies. These operations constitutes the computationally most expensive part of the signing procedure in SQISIGN.

2.6.1. The correspondence between ideals and isogenies

Given an elliptic curve E, an *endomorphism of* E is an isogeny $\varphi : E \to E$. The collection of all endomorphisms of E is called the *endomorphism ring* of E, written $\mathrm{End}(E)$. If we now let E be a supersingular elliptic curve over $\mathbb{F}_{p^2}$, $\mathrm{End}(E)$ is isomorphic to a maximal order $\mathcal{O}$ in the quaternion algebra $B_{p,\infty}$. Fixing an isomorphism $\mathcal{O} \cong \mathrm{End}(E)$, an element $\alpha \in \mathcal{O}$ corresponds to an endomorphism of E, and we write, by slight abuse of notation, $\alpha(P)$ for $P \in E$ to denote the image of α under a (fixed) isomorphism evaluated at $P \in E$, and similarly, we write $\ker \alpha$ to denote the kernel of the image of α.

Given the curve E, the order $\mathcal{O}$, and an isomorphism $\mathcal{O} \cong \mathrm{End}(E)$, we obtain a bijection between:

- left ideals of $\mathcal{O}$ of norm coprime to p, and
- finite subgroups of E (kernels of separable isogenies from E).

Explicitly, this correspondence is given by sending an ideal I to the finite subgroup

$$E[I] := \{P \in E \mid \alpha(P) = \infty, \forall \alpha \in I\},$$

which, when I is written as $I = \mathcal{O}\langle \alpha, N \rangle$, simplifies to

$$E[I] := \ker \alpha \cap E[N]$$

Given such an I, we denote the corresponding separable isogeny with kernel $E[I]$ by φ_I. Reciprocally, given a separable isogeny φ from E, we denote the corresponding ideal by I_φ, which can explicitly be given as

$$I_\varphi = \{\alpha \in \mathcal{O} \mid \alpha(P) = \infty, \forall P \in \ker \varphi\}$$

In SQISIGN, we will always use a prime $p \equiv 3 \pmod 4$. In this case, the curve

$$E_0 : y^2 = x^3 + x$$

is supersingular, with endomorphism ring isomorphic to

$$\mathcal{O}_0 \;=\; \mathbb{Z} \oplus i\,\mathbb{Z} \oplus \frac{i+j}{2}\,\mathbb{Z} \oplus \frac{1+k}{2}\,\mathbb{Z}$$

Algorithm 17 SigningKLPT$_{2^e}(K, I_\tau)$

Input: K a left $\mathcal{O}$-ideal
Input: I_τ a $(\mathcal{O}_0, \mathcal{O})$-ideal of prime norm N_τ, coprime to $\mathrm{nrd}(K)$
Output: Generator β of equivalent ideal $J \sim I$ of norm 2^e
Output: `found` a boolean indicating whether a solution was found

$L :=$ RandomEquivalentPrimeIdeal(K)
$L' := [I_\tau]^* L$
$I :=$ RandomEquivalentPrimeIdeal(L') and let δ be such that $I = \chi_{L'}(\delta)$
$N_I := \mathrm{nrd}(I)$
$k := \lceil \log_2(p) \rceil - \lfloor \log_2(N_I) \rfloor +$ `KLPT_gamma_exponent_center_shift`
$N_\gamma := 2^k$
$N_\mu := 2^{e-k}$
Initialise `counter` := 0 and `found` := False
while not `found` and `counter` $<$ `KLPT_signing_num_gamma_trial` **do**
  `counter` := `counter` $+ 1$
  γ, `found` := FullRepresentInteger$(N_I N_\gamma)$
  if not `found` **then**
    Continue
  end if
  $C_0, D_0 :=$ EichlerModConstraint$(I, \gamma, 1,$ True$)$
  Compute $\alpha_0 := j(C_0 + iD_0)$
  if $N_\mu / \mathrm{nrd}(\alpha_0)$ is not a square $\bmod N_I$ **then**
    Continue
  end if
  Set λ_0 to be the square root of $N_\mu / \mathrm{nrd}(\alpha_0) \bmod N_I$
  $C_1, D_1 :=$ EichlerModConstraint$(I_\tau, \gamma, \delta,$ False$)$
  Compute $\alpha_1 := j(C_1 + iD_1)$
  if $N_\mu / \mathrm{nrd}(\alpha_1)$ is not a square $\bmod N_\tau$ **then**
    Continue
  end if
  Set λ_1 to be the square root of $N_\mu / \mathrm{nrd}(\alpha_0) \bmod N_\tau$
  $\lambda := 2 \cdot \mathsf{CRT}_{N_I, N_\tau}(\lambda_0, \lambda_1)$
  $N_\mu := 4 N_\mu$
  $C := \mathsf{CRT}_{N_I, N_\tau}(C_0, C_1)$, $D := \mathsf{CRT}_{N_I, N_\tau}(D_0, D_1)$
  `max` := `KLPT_signing_number_strong_approx`
  μ, `found` := FullStrongApproximation$_{N_\mu}(\mathcal{O}, N_I N_\tau, C, D, \lambda,$ `max`$)$
  if $\mathrm{tr}(\gamma\mu\delta) \equiv 0 \mod 2$ **then**
    `found` := False
  end if
end while
return $\gamma\mu$, `found`

The isomorphism is given by sending j to the Frobenius endomorphism $(x, y) \to (x^p, y^p)$, and i to the automorphism $(x, y) \to (-x, \sqrt{-1}y)$ on E_0. For the remainder of this document, we fix this choice of $E_0, \mathcal{O}_0$ and isomorphism $\mathrm{End}(E_0) \cong \mathcal{O}_0$, while $\mathcal{O}$ refers to an arbitrary maximal order (not necessarily distinct from $\mathcal{O}_0$). Whenever we are given an isogeny $\varphi : E_0 \to E$ and the corresponding $(\mathcal{O}_0, \mathcal{O})$-ideal I, the isomorphism $\mathcal{O}_0 \cong \mathrm{End}(E_0)$ naturally induces an isomorphism $\mathcal{O} \cong \mathrm{End}(E)$. In the following algorithms, we will either work with E_0, or with some E for which we know such a pair (φ, I). So we do not explicitly mention which isomorphism we use for the correspondence: we use the implicitly defined isomorphism $\mathcal{O} \cong \mathrm{End}(E)$.

2.6.2. Converting $\mathcal{O}_0$-ideals to isogenies

The simplest translation of ideals to their corresponding isogeny happens for $\mathcal{O}_0$-ideals, whose corresponding kernel is can be defined over $\mathbb{F}_{p^2}$, by using both quadratic twists of E_0 (i.e. ideals whose norm divides p^2-1). We depict this in Algorithm 18. When translating the ideal $I := \mathcal{O}_0\langle \alpha, D\rangle$, the algorithm works by finding the action of

$$\alpha = x_1 + x_2 i + x_3 \frac{i+j}{2} + x_4 \frac{1+k}{2}, \;\; x_i \in \mathbb{Z}\,.$$

on a fixed basis of $P_D, Q_D \in E[D]$, where the action of i, $\frac{i+j}{2}$, $\frac{1+k}{2}$ are precomputed, and recovering $\ker \alpha \cap E[D]$. Explicitly, one can recover the action $\mathbf{M}_\alpha \in \mathbb{M}_2(\mathbb{Z}/D\,\mathbb{Z})$ as

$$\mathbf{M}_\alpha = x_1\mathbf{I} + x_2\mathbf{M}_i + x_3\mathbf{M}_{\frac{i+j}{2}} + x_4\mathbf{M}_{\frac{1+k}{2}}\,,$$

before computing some vector $[a,b]^T \in \ker \mathbf{M}_\alpha$, which gives the corresponding generator $[a]P_D + [b]Q_D$ of the desired isogeny.

Outputting the push-through isogeny. In SQISIGN, Algorithm 18 is used as a subroutine in $\mathsf{IdealToIsogenyEichler}_{\ell^\bullet}$, and when computing the commitment isogeny. In both these cases, we are typically interested in computing the push-through $[\varphi]_*\varphi_I$, for some isogeny φ of coprime degree (i.e. the isogeny with kernel $\varphi(\ker \varphi_I)$).

This push-through comes for free by directly applying the coefficients found to the basis $P'_D, Q'_D := \varphi(P_D), \varphi(Q_D)$. This comes from the fact that

$$\varphi(\ker \varphi_I) = \varphi([a]P_D + [b]Q_D) = [a]\varphi(P_D) + [b]\varphi(Q_D) = [a]P'_D + [b]Q'_D.$$

Because of this, Algorithm 18 takes in a basis P'_D, Q'_D of $E[D]$ (where E may be distinct from E_0), where this basis is the image of the fixed D-torsion basis on E_0 under some isogeny φ of coprime degree to φ_I.

Kernel to isogeny. In Algorithm 18 and several other places throughout this document, we will denote by KernelToIsogeny a function which takes as input an elliptic curve point, and outputs the separable isogeny, whose kernel is generated by that point, for instance using the appropriate choice of Vélu or $\sqrt{\text{élu}}$ (see Section 2.3).

Algorithm 18 $\mathsf{IdealToIsogeny}_D(I, P'_D, Q'_D)$

Input: I a left $\mathcal{O}_0$-ideal of norm dividing D
Input: P'_D, Q'_D the image of the precomputed basis of $E[D]$, under some isogeny φ
Output: $[\varphi]_*\varphi_I$

1: Compute α such that $I = \mathcal{O}_0\langle \alpha, \mathrm{nrd}(I)\rangle$
2: Let $\mathbf{A} = [1, i, \frac{i+j}{2}, \frac{1+k}{2}]$ denote a basis of $\mathcal{O}$
3: Compute $\mathbf{v}_\alpha := [x_1, x_2, x_3, x_4]^T \in \mathbb{Z}^4$ such that $\mathbf{A}\mathbf{v}_\alpha = \alpha$
4: $\mathbf{M}_\alpha := x_1\mathbf{I} + x_2\mathbf{M}_i + x_3\mathbf{M}_{\frac{i+j}{2}} + x_4\mathbf{M}_{\frac{1+k}{2}}$
5: Compute $[a,b]^T \in \ker \mathbf{M}_\alpha$
6: $K := [a]P'_D + [b]Q'_D$
7: **return** KernelToIsogeny(K)

2.6.3. Converting $\mathcal{O}$-ideals of large prime power norm

In the key generation and signing procedure of SQISIGN, the ideal that is translated to the signature isogeny is an $\mathcal{O}$-ideal of large ℓ-power norm. This translation happens using $\mathsf{IdealToIsogenyEichler}_{\ell^\bullet}$ (Algorithm 19) to translate such an ideal to its corresponding isogeny. This algorithm has of two main phases; one consisting of purely quaternion operations, and one for doing the elliptic curve operations. The goal of the first phase is to compute the quaternions $\beta_i \in \mathcal{O}_i$ (using SpecialEichlerNorm, Algorithm 15), needed for the isogeny translation in phase two. From each step of phase one, we save the ideals L_i, R_i, encoding information about β_i, as well as the integer values C_i, D_i, and t_i, the reduced trace of β_i. Phase two translates the ideals I_i to their corresponding isogenies, by using the ideals L_i, R_i, and the values C_i, D_i, t_i saved during phase one. Figure 2 shows how the involved ideals relate.

Algorithm 19 $\mathsf{IdealToIsogenyEichler}_{\ell^\bullet}(I, J, B_{A,T}, Q)$

Input: I a left $\mathcal{O}$-ideal of norm ℓ^e, where $e = fg$
Input: J a $(\mathcal{O}_0, \mathcal{O})$-ideal of norm $\ell^\bullet$
Input: $B_{A,T}$, a basis of $E_A[T]$
Input: Q, a point of $E_A[\ell^f]$
Output: φ_I corresponding to I
Output: $\mathbf{zip} = (b, s_1, s_2, \ldots, s_g)$ compressed φ_I
Output: `found` a boolean indicating that the computation was successful

Set $J_1 := J, I_1 := I + \ell^f\mathcal{O}, I_1' := I_1^{-1}I, \mathcal{O}_1 := \mathcal{O}, K_1 := \bar{J} + \ell^f\mathcal{O}$
for $i \in \{1, \ldots, g\}$ **do**
  $L_i, R_i, C_i, D_i, t_i, \texttt{found} := \mathsf{IdealStep}(I_i, K_i, J_i)$
  if not `found` **then**
    return _, _, `found`
  end if
  $J_{i+1} := J_i I_i$
  $K_{i+1} := \bar{I}_i$
  $\mathcal{O}_{i+1} := \mathcal{O}_L(I_i')$
  $I_{i+1} := I_i' + \ell^f\mathcal{O}_{i+1}$
  $I_{i+1}' := I_{i+1}^{-1} I_i'$
end for
Set $Q_{\ell^f} := Q$
Let $P_T, Q_T := B_{A,T}$
$\varphi_I := [1] \in \mathrm{End}(E_A)$
$\varphi_{I_i}, Q_{\ell^f}, s_1, b := \mathsf{IsogenyStep}(L_i, R_i, C_i, D_i, t_i, Q_{\ell^f}, P_T, Q_T, \textsc{True})$
for $i \in \{2, \ldots, g\}$ **do**
  $P_T, Q_T := \varphi_{I_{i-1}}(P_T), \varphi_{I_{i-1}}(Q_T)$
  $\varphi_I := \varphi_{I_{i-1}} \circ \varphi_I$
  $\varphi_{I_i}, Q_{\ell^f}, s_i, \_ := \mathsf{IsogenyStep}(L_i, R_i, C_i, D_i, t_i, Q_{\ell^f}, P_T, Q_T, \textsc{False})$
end for
$\varphi_I := \varphi_{I_g} \circ \varphi_I$
$\mathbf{zip} := (b, s_1, s_2, \ldots, s_g)$
return φ_I, **zip**, `found`

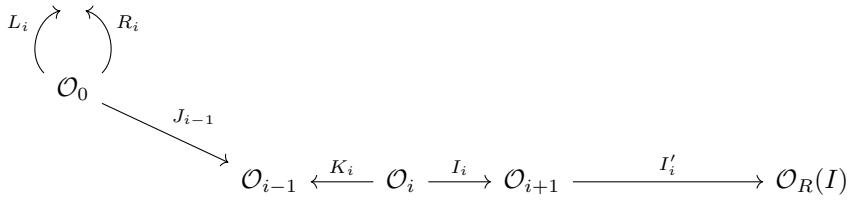

FIGURE 2. The ideals involved in each iteration the first phase of $\mathsf{IdealToIsogenyEichler}_{\ell^\bullet}$. Note that $J_i = J_{i-1}\bar{K}_i$. Further, L_i and R_i are ideals of norm dividing T, J_i and I_i' are ideals of norm $\ell^\bullet$, while K_i and I_i are ideals of norm ℓ^f.

Choice of ℓ. In verification, we will need to compute chains of ℓ-isogenies, and therefore we choose $\ell = 2$ to obtain efficient verification. In the search for SQISIGN friendly primes p, described in Chapter 5, this means we require a large power of 2 to divide $p^2 - 1$

Handling Failures. $\mathsf{IdealToIsogenyEichler}_{\ell^\bullet}$ may fail as a result of $\mathsf{SpecialEichlerNorm}_T$ failing to find a solution. Because of this, the algorithm does all necessary executions of $\mathsf{SpecialEichlerNorm}_T$ first, before doing any costly elliptic curve operations. In case of failure, the signing procedure must be restarted.

Algorithm 20 $\mathsf{IdealStep}(I, K, J)$

Input: I a left $\mathcal{O}$-ideal of norm ℓ^f
Input: K a left $\mathcal{O}$-ideal of norm ℓ^f
Input: J an $(\mathcal{O}_0, \mathcal{O})$-ideal of norm $\ell^\bullet$
Output: L, R left $\mathcal{O}_0$-ideals of norm dividing T with $O_R(R) = O_R(L)$
Output: $C, D \in \mathbb{Z}$
Output: `found` a boolean indicating that a solution was found

$\beta, \texttt{found} := \mathsf{SpecialEichlerNorm}_T(\mathcal{O}, K + \ell\mathcal{O})$
$C, D := \mathsf{FindLinearCombination}(\beta, I, K)$
$t := \mathrm{trd}(\beta)$
Write $\mathrm{nrd}(\beta) = n_1 n_2$, with $n_i \mid T$ and $n_2 \mid T$
$H_1 := \mathcal{O}_i\langle \beta_i, n_1 \rangle$
$H_2 := \mathcal{O}_i\langle \bar{\beta}_i, n_2 \rangle$
$L := [J_i]^* H_1$
$R := [J_i]^* H_2$
return L, R, C, D, t, `found`

Algorithm 21 $\mathsf{IsogenyStep}(L, R, C, D, t, Q_{\ell^f}, P_T, Q_T, \mathsf{FirstStep})$

Input: L, R left $\mathcal{O}_0$-ideals of norm dividing T with $O_R(R) = O_R(L)$
Input: $C, D \in \mathbb{Z}$
Input: t the trace of β
Input: Q_{ℓ^f} a point of order ℓ^f (generating the dual of the previous isogeny)
Input: P_T, Q_T a T-torsion basis
Input: FirstStep a boolean to indicate the first step
Output: φ generated by $P + [s]Q$
Output: Q', a generator of $\widehat{\varphi}$
Output: s a scalar s expressing the kernel generator as a linear combination of a deterministically generated basis
Output: A bit b, indicating a swapped basis

$(Q_{\ell^f}, P_{\ell^f}) := \mathsf{CompleteBasis}_{\ell^f, p+1}(E, Q_{\ell^f})$
$M_\beta := \mathsf{EndomorphismAction}(L, R, t, P_{\ell^f}, Q_{\ell^f}, P_T, Q_T)$
$[a_0, a_1]^T := [C, 0]^T + M_\beta [D, 0]^T$
$G := [a_0]P_{\ell^f} + [a_1]Q_{\ell^f}$
$b := 0$
if FirstStep: **then**
  $(Q'_{\ell^f}, P'_{\ell^f}) := \mathsf{TorsionBasis}_{\ell^f, p+1}(E)$
  $a_0, a_1 := \mathsf{NormalizedDlog}_{\ell^f}(E, (P'_{\ell^f}, Q'_{\ell^f}), G)$
  if $a_0 \equiv 0 \pmod{\ell}$ **then**
    Swap P'_{ℓ^f} and Q'_{ℓ^f}
    Swap a_0 and a_1
    $b := 1$
  end if
end if
$s := a_0^{-1} a_1 \pmod{\ell^f}$
$\varphi := \mathsf{KernelToIsogeny(G)}$
$Q' := \varphi(Q)$
return φ, Q', s, b

Compression. The compression of the output isogeny φ_I is computed together with φ_I. See Paragraph 3.5 for more detailed description, as well as the decompression procedure.

Endomorphism Evaluation. As part of the subroutine IsogenyStep, one needs to evaluate $\beta(P_\ell)$, for some point $P_\ell \in \ker \varphi_K$, from the isogenies φ_1, φ_2 satisfying $\beta = \widehat{\varphi_2} \circ \varphi_1$. A naïve way to do this is to simply push P_{ℓ^f} through $\widehat{\varphi_2} \circ \varphi_1$. While this works, it is very expensive to compute $\widehat{\varphi_2}$ from φ_2, since φ_2 will be of degree T. To avoid this, note that solving

$$\widehat{\varphi_2} \circ \varphi_1(P_{\ell^f}) = [x_1']P_{\ell^f} + [x_2']Q_{\ell^f}$$

is essentially the same as solving

$$\varphi_1(P_{\ell^f}) = [x_1]\varphi_2(P_{\ell^f}) + [x_2]\varphi_2(Q_{\ell^f}),$$

up to multiplying by $\deg \varphi_2$. Hence, we can avoid computing the dual by instead directly computing the decomposition of $\widehat{\varphi_2} \circ \varphi_1(P_{\ell^f})$ on a fixed basis. To do this, we need to make sure the codomain of ϕ_1 and ϕ_2 are equal (not just isomorphic). This can be done by using the implicit function Canonical, see Section 3.3 for more details.

An additional complication comes from sign ambiguity which arises from the definition of NormalizedDlog (Algorithm 4). This can be fixed by recovering the whole action of β, and checking that the trace is correct (see Algorithm 22).

Algorithm 22 $\mathsf{EndomorphismAction}_{\ell^f}(L, R, t, P_{\ell^f}, Q_{\ell^f}, P_T, Q_T)$

Input: L, R left $\mathcal{O}_0$-ideals of norm dividing T with $O_R(R) = O_R(L)$
Input: t trace of β
Input: P_{ℓ^f}, Q_{ℓ^f} ℓ^f-torsion basis on E
Input: P_T, Q_T T-torsion basis on E
Output: M_β the matrix corresponding to the action of β (up to some invertible scalar multiple)

1: $\varphi_1 : E \to F := \mathsf{IdealToIsogeny}_T(L, P_T, Q_T)$
2: $\varphi_2 : E \to F' := \mathsf{IdealToIsogeny}_T(R, P_T, Q_T)$
3: Compute the isomorphism $\omega : F \to F'$.
4: $x_1, x_2 := \mathsf{NormalizedDlog}_{\ell^f}(F', (\varphi_2(P_{\ell^f}), \varphi_2(Q_{\ell^f})), \omega \circ \varphi_1(P_{\ell^f}))$
5: $x_3, x_4 := \mathsf{NormalizedDlog}_{\ell^f}(F', (\varphi_2(P_{\ell^f}), \varphi_2(Q_{\ell^f})), \omega \circ \varphi_1(Q_{\ell^f}))$
6: Try $a, b \in \{1, -1\}$ until $(ax_1 + bx_4) \deg \varphi_2 \equiv t \pmod{\ell^f}$.
7: $\mathbf{M}_\beta := \begin{pmatrix} ax_1 & ax_2 \\ bx_3 & bx_4 \end{pmatrix}$
8: **return** $\mathbf{M}_\beta$

2.6.4. Converting isogenies to ideals

Translating isogenies to their corresponding ideals is comparatively more straightforward. To find the ideal corresponding to the isogeny generated by a point P of order D is to create a basis $\langle P, \theta(P) \rangle = E[D]$ for some endomorphism θ (often called a distortion map), and decomposing $\eta(P)$ (where η is orthogonal to θ) along this basis, i.e. solving

$$[a]P + [b]\theta(P) = \eta(P),$$

which means that $\alpha = a + b\theta - \eta$ sends the point P to the identity, and hence $I = \mathcal{O}\langle \alpha, D \rangle$ is the corresponding ideal.

Using a decomposition along a fixed basis. In SQISIGN, we only ever translate points on E_0 to their corresponding ideals, hence we fix the choice $\theta = j + \frac{1+k}{2}$ and $\eta = i$. Further, instead of taking as input a kernel point P, KernelDecomposedToIdeal (Algorithm 23) takes in the decomposition of P along a fixed, precomputed basis, i.e. $P = [a]P_D + [b]Q_D$. By having again precomputed the action of θ and η on this basis (similar to the simplest ideal to isogeny translation), we obtain the required ideal without requiring any discrete log computations.

Algorithm 23 $\mathsf{KernelDecomposedToIdeal}_D(c_1, c_2)$

Input: $a, b \in \mathbb{Z}$ defining a point $[a]P_D + [b]P_D$ on E_0 of order D generating an isogeny ϕ
Output: I_ϕ a left $\mathcal{O}_0$-ideal
1: Set $\mathbf{v}_1 := [c_1, c_2]^T$
2: $\mathbf{v}_2 := \mathbf{M}_\theta \mathbf{v}_1$
3: $\mathbf{M} := [\mathbf{v}_1, \mathbf{v}_2] \in \mathbb{M}_{2\times 2}(\mathbb{Z}/D\,\mathbb{Z})$
4: $[a, b]^T := \mathbf{M}^{-1}\mathbf{M}_i\mathbf{v}_1$
5: $I := \mathcal{O}_0\langle a + b(j + \frac{1+k}{2}) - i, D\rangle$
6: **return** I

CHAPTER 3

Signature

In this chapter we describe the key generation, signing and verification procedure in SQISIGN. At the end, we list the concrete parameter sets implemented in SQISIGN for all three security levels.

3.1. Σ-protocols and the Fiat–Shamir Heuristic

Our signature scheme SQISIGN is constructed from a Σ-protocol using the Fiat–Shamir heuristic. For completeness, we recall the relevant definitions, which will also be used to prove the security of SQISIGN in Chapter 9.

Σ-protocols. We begin by defining a Σ-protocol. Recall that a Σ-protocol for an NP-language $\mathcal{L}$ is a public-coin three-move interactive proof system consisting of two parties: a verifier and a prover. The prover is given a witness w for an element $x \in \mathcal{L}$, their goal is to convince the verifier that they know w.

DEFINITION 3.1.1 (Σ-protocol). A Σ-protocol Π_Σ for a family of relations $\{\mathcal{R}\}_\lambda$ parameterized by security parameter λ consists of PPT algorithms $(\mathsf{P}_1, \mathsf{P}_2, \mathsf{V})$. Throughout this section, we assume that the prover algorithms P_1 and P_2 share state (and hence avoid explicitly passing state between these algorithms). We also assume throughout that the final verification algorithm V is deterministic. The protocol proceeds as follows:

(1) The prover, on input $(x, w) \in \mathcal{R}$, returns a commitment $\mathsf{com} \leftarrow \mathsf{P}_1(x, w)$ which is sent to the verifier.
(2) The verifier uniformly randomly samples a challenge string $\mathsf{chall} \leftarrow \{0,1\}^\lambda$ and sends the challenge to the prover.
(3) On receiving the challenge chall from the verifier, the prover runs $\mathsf{resp} = \mathsf{P}_2(\mathsf{chall})$ and returns the response resp to the verifier.
(4) The verifier runs the final verification algorithm $\mathsf{V}(x, \mathsf{com}, \mathsf{chall}, \mathsf{resp})$ and outputs a bit $b \in \{0,1\}$.

We refer to the tuple $(\mathsf{com}, \mathsf{chall}, \mathsf{resp})$ as a **transcript** of the Σ-protocol. A transcript $(\mathsf{com}, \mathsf{chall}, \mathsf{resp})$ is said to be valid, or accepting, if $\mathsf{V}(x, \mathsf{com}, \mathsf{chall}, \mathsf{resp})$ outputs 1. Certain desirable properties of a Σ-protocol include: (1) correctness, if a prover knows $(x, w) \in \mathcal{R}$ and behaves honestly, then the verifier will output 1; (2) special soundness, if a cheating prover only knows statement x and not a corresponding witness w, they cannot force an honest verifier to accept the proof; (3) honest-verifier zero-knowledge, no malicious verifier cannot extract additional knowledge from the prover. These are defined in more detail in Chapter 9.

The Fiat-Shamir Transform. We now describe the standard transformation from a Σ-protocol $(\mathsf{P}_1, \mathsf{P}_2, \mathsf{V})$ with a random instance generator Gen into a digital signature scheme $\Gamma = (\mathsf{Gen}_{\mathsf{Sig}}, \mathsf{Sign}, \mathsf{Ver})$ via the Fiat–Shamir transform [**FS87**]. The transformation uses a hash function $H : \{0,1\}^* \to \{0,1\}^\lambda$ (modeled as a random oracle) and works as follows.

- $\mathsf{Gen}_{\mathsf{Sig}}(1^\lambda)$: Generate $(x, w) \leftarrow \mathsf{Gen}(1^\lambda)$. Output the signing key $\mathsf{sk} = (x, w)$ and the verification key $\mathsf{pk} = x$.
- $\mathsf{Sign}(\mathsf{sk}, \mathsf{msg})$: On input the signing key $\mathsf{sk} = (x, w)$ and a message $\mathsf{msg} \in \{0,1\}^*$, sample
$$\mathsf{com} \leftarrow \mathsf{P}_1(x, w), \quad \mathsf{chall} = H(\mathsf{com} \,\|\, \mathsf{msg}), \quad \mathsf{resp} = \mathsf{P}_2(\mathsf{chall}),$$
and output the signature $\sigma = (\mathsf{com}, \mathsf{resp})$.
- $\mathsf{Ver}(\mathsf{pk}, \mathsf{msg}, \sigma)$: On input the verification key $\mathsf{pk} = x$, a message $\mathsf{msg} \in \{0,1\}^*$ and a signature $\sigma = (\mathsf{com}, \mathsf{resp})$, output 1 (i.e., valid) if for $\mathsf{chall} = H(\mathsf{com} \,\|\, \mathsf{msg})$
$$\mathsf{V}\,(x, \mathsf{com}, \mathsf{chall}, \mathsf{resp}) = 1,$$

else output 0 (i.e., invalid).

To highlight the connection with an interactive Σ-protocol, we will describe SQIsign signature in terms of the three stages: commitment, challenge, and response.

3.2. Precomputation

The SQIsign algorithm assumes some precomputed data given the global parameter p, a prime determined as described in Chapter 5. The precomputed values include:

- The odd, smooth value T determining the "accessible torsion", satisfying $T \mid p^2 - 1$, and $T > p^{5/4}$.
- The largest integer f such that $2^f \mid p + 1$.
- The largest integer g such that $3^g \mid T$.
- The degree of the commitment isogeny D_{com}, satisfying $D_{\text{com}} \mid T$. For the provided parameter sets, we define it as $D_{\text{com}} = T/3^g$.
- The degree of the challenge isogeny D_{chall}, satisfying $D_{\text{chall}} \mid T$ and $\gcd(D_{\text{com}}, D_{\text{chall}}) = 1$. For the provided parameter sets, we define it as $D_{\text{chall}} = 2^f 3^g$.
- $B_{0,T} = [P_T, Q_T]$, a basis of $E_0[T]$.
- $B_{0,T} = [P_{D_{\text{com}}}, Q_{D_{\text{com}}}]$, a basis of $E_0[D_{\text{com}}]$.
- $B_{0,T} = [P_{D_{\text{chall}}}, Q_{D_{\text{chall}}}]$, a basis of $E_0[D_{\text{chall}}]$.
- The action of $\theta = j + (1 + k)/2$ on $B_{0,D_{\text{com}}}$.
- The action of $\alpha \in \{i, j, k, \frac{i+j}{2}, \frac{1+k}{2}\}$ on $B_{0,T}$ and $B_{0,D_{\text{chall}}}$.
- A list `extremal_order_list` of alternate p-extremal maximal orders.
- Constants required by the various quaternion-related algorithms (more details on these constants is given in Section 7).

Torsion on curve and on twist. In practice, SQIsign uses x-only arithmetic to avoid going to higher extension fields than $\mathbb{F}_{p^2}$. However, this also means that one has to split the torsion basis up in a part defined over the curve itself, and its twist. Specifically, let $T = T^+T^-$, where $T^\pm \mid p \pm 1$. We compute and store a basis $B_{0,T^+} = \langle P_{0,T^+}, Q_{0,T^+} \rangle$ of order T^+ on E_0 and a basis $B_{0,T^-} = \langle P_{0,T^-}, Q_{0,T^-} \rangle$ of order T^- on its twist. All computations requiring the T-torsion basis, must then be split into a computation on both the T^+ and T^- basis. This includes the relevant precomputation, such as computing the action of certain endomorphisms (see Section 2.6) on the T-torsion basis. The same happens for D_{com}. However, for ease of exposition, we only refer to the T-torsion basis in this (and the previous) chapter.

3.3. Key generation

The key generation algorithm SQIsign.KeyGen (Algorithm 25), takes input 1^λ (where λ is the security parameter), and outputs a secret signing key `sk` and public verification key `pk`. At a high level, the goal of the key generation is to produce the public key, a random curve E_A, and the secret key, data required compute with $\mathrm{End}(E_A)$.

For efficiency, the secret key includes some additional data (technically not necessary to recover $\mathrm{End}(E_A)$) that will be useful to generate signatures. More concretely, the secret key contains :

(1) a quaternion element α, generating the two ideals $I_{\text{secret}}, J_{\text{secret}}$ corresponding to the secret key isogenies.
(2) a basis $B_{A,T}$, which is the image of the basis $B_{0,T}$ through an isogeny $\varphi_{\text{secret}} : E_0 \to E_A$ of degree $2^\bullet$.
(3) A point $Q \in E_A[2^f]$ corresponding to the dual of the kernel of the last 2^f isogeny composing φ_{secret}.

More specifically, the key generation proceeds by first sampling a random secret left $\mathcal{O}_0$-ideal I_{secret} of secret prime norm $D_{\text{secret}} \equiv 3 \bmod 4$ of size $\approx p^{1/4}$. This ideal will be used to compute the input to SigningKLPT in the signing procedure. Subsequently, the key generation computes an equivalent ideal $J_{\text{secret}} \sim I_{\text{secret}}$ of norm $2^\bullet$ (using KeyGenKLPT), before translating J_{secret} to its corresponding isogeny $\varphi_{\text{secret}} : E_0 \to E_A$, using $\mathsf{IdealToIsogenyEichler}_{2^\bullet}$. The ideal J_{secret}, and corresponding isogeny φ_{secret}, as well as the image $B_{A,T}$ of the precomputed basis $B_{0,T}$ under φ_{secret} and the kernel of the dual of the last step of φ_{secret} are all used to translate the output of SigningKLPT in the signing procedure to its corresponding isogeny.

Sampling random ideals. In the key generation procedure of SQIsign, sampling the random secret ideal is done by first sampling a random secret prime $D_{\text{secret}} \equiv 3 \bmod 4$ of size $\approx p^{1/4}$. One then finds $\gamma \in \mathcal{O}_0$ of norm $D_{\text{secret}} 2^{\lceil \log_2(p) \rceil}$ using FullRepresentInteger. Finally, sampling a random scalar $0 < a < D_{\text{secret}}$, one can output the ideal $\mathcal{O}_0\langle \gamma(a + i), D_{\text{secret}} \rangle$. Note that $\mathrm{nrd}(a + i) = a^2 + 1$, which cannot be divisible by D_{secret} since $D_{\text{secret}} \equiv 3 \pmod 4$, hence the output ideal does indeed correspond to an isogeny of degree D_{secret}.

Handling failures. As noted in Chapter 2, several procedures used in SQIsign.KeyGen and SQIsign.Sign may fail. The failures are handled by retrying with different randomness. The number of tries are bounded by two constant `SQIsign_keygen_attempts` and `SQIsign_response_attempts`.

Deterministic choice of Montgomery coefficient. The correspondence between ideals and isogenies (see Section 2.6) is only defined up to isomorphism. Because different ways of computing isogenies may lead to different Montgomery equations, MontgomeryNormalize is used to define models that only depend on the isomorphism class. We extend this normalization to isogenies through Algorithm Normalized.

Algorithm 24 Normalized(ϕ)

Input: An isogeny $\phi : E_1 \to E_2$
Output: A normalized curve E_2' isomorphic to E_2
Output: An isogeny $\phi' : E_1 \to E_2'$ equal to ϕ up to post-composition by an isomorphism.
1: $E_2', \iota :=$ MontgomeryNormalize(E_2)
2: **return** $E_2', \iota \circ \phi$

Algorithm 25 SQIsign.KeyGen(1^λ)

Input: 1^λ where λ is the security parameter
Output: Secret signing key `sk` and public verification key `pk`
Output: `found` a boolean indicating whether computation succeeded
1: Set `found` := FALSE `counter := 0`
2: **while** `found` = FALSE and `counter` $<$ `SQISIGN_keygen_attempts` **do**
3: `counter` := `counter` $+ 1$
4: Select a random `KLPT_secret_key_prime_size`-bit prime $D_{\text{secret}} \equiv 3 \bmod 4$
5: Set I_{secret} to be a random ideal of norm D_{secret}
6: α, `found` := KeyGenKLPT$_{2^\bullet}(I_{\text{secret}})$
7: **if** not `found` **then**
8: CONTINUE
9: **end if**
10: $J_{\text{secret}} := \chi_{I_{\text{secret}}}(\alpha)$
11: $\varphi_{\text{secret}}, \_,$ `found` := IdealToIsogenyEichler$_{2^\bullet}(J_{\text{secret}}, \mathcal{O}_0, B_{0,T})$
12: **end while**
13: **if** `found` **then**
14: $\alpha := \bar{\alpha}$
15: $E_A, \varphi_{\text{secret}} :=$ Normalized$(\varphi_{\text{secret}})$
16: $B_{A,T} := \varphi_{\text{secret}}(B_{0,T})$
17: Let P be a point generating $\ker \varphi_{\text{secret}} \cap E_0[2^f]$
18: $(P, Q) :=$ CompleteBasis$_{2^f, p+1}(E_0, P)$
19: $Q := \varphi_{\text{secret}}(Q)$
20: Set `pk` := E_A
21: Set `sk` := $\big(\alpha, B_{A,T}, Q\big)$
22: **end if**
23: **return** `sk`, `pk`, `found`

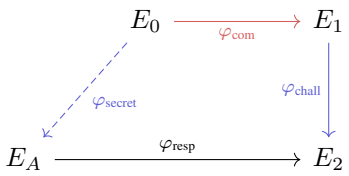

FIGURE 1. The SQIsign protocol with three phases: commitment φ_{com}, challenge φ_{chall} and response φ_{resp}.

3.4. Signing

The SQISIGN signing algorithm can naturally be divided into three stages: commitment phase, challenge phase and the response phase. To apply this transformation, one applies an appropriate hash function H, which takes as input a message msg and a normalized Montgomery curve E, and outputs a kernel generator $K_{\text{chall}} \in E$ of order D_{chall}.

This signing procedure SQIsign.Sign (Algorithm 26) takes as input a secret key $\mathtt{sk}$ and message msg, and outputs a signature σ. At a high level, the commitment phase consists of computing a random isogeny $\varphi_{\text{com}} : E_0 \to E_1$, and the ideal I_{com}. The challenge phase consists of hashing the commitment E_1 and the message to a point generating the challenge isogeny $\varphi_{\text{chall}} : E_1 \to E_2$, and computing the ideal I_{chall}. Finally, the response phase consists of computing an equivalent ideal $J \sim \overline{I_{\text{secret}}} \cdot I_{\text{com}} \cdot I_{\text{chall}}$, and translating J to its corresponding isogeny ϕ_J. Below, we give more detail on each phase of the signing procedure.

Commitment. The commitment starts by sampling a random commitment ideal I_{com} of norm D_{com} and then computing the isogeny $\varphi_{\text{com}} : E_0 \to E_1$ corresponding to ideal I_{com}. The commitment isogeny has to be composed with an isomorphism to the normalized Montgomery curve for the isomorphism class of E_1. Finally, the commitment phase ends with pushing the basis $B_{0,D_{\text{chall}}}$ through this composition, to obtain P_1, Q_1.

Challenge. The challenge phase starts by hashing the message msg and normalized Montgomery curve E_1 to obtain the kernel K_{chall} of the challenge isogeny $\varphi_{\text{chall}} : E_1 \to E_2$. We define the hash function H as follows: on input a message msg and a curve E_1, compute $a = \mathtt{SHAKE256}(\mathsf{msg} \,||\, j(E_1))$. Then, compute R_1, S_1 a deterministic basis of $E_1[D_{\text{chall}}]$ and set the output $K_{\text{chall}} = R_1 + [a]S_1$.

Similarly to φ_{com}, φ_{chall} also needs to be composed with an isomorphism to the normalized Montgomery curve corresponding to the isomorphism class of E_2. K_{chall} is then decomposed along the basis P_1, Q_1 obtained during the commitment phase. From this decomposition, we obtain the corresponding ideal I_{chall} (by using KernelDecompositionToIdeal). Next, the commitment computes a compression of the dual of the challenge isogeny. This is done by computing a generator P' of $\ker \widehat{\varphi_{\text{chall}}}$, before decomposing this along a deterministically computed D_{chall}-torsion basis, on the normalized Montgomery curve E_2. Finally, the challenge phase ends with deterministically computing a generator Q of φ_{chall} (essentially by computing a generator of the dual of the dual of φ_{chall}), and computing a scalar r, such that $[r]Q = K_{\text{chall}}$.

Response. The response phase consists of first finding an equivalent ideal $J \sim \overline{I_{\text{secret}}} \cdot I_{\text{com}} \cdot I_{\text{chall}}$ of $2^\bullet$ norm, using SigningKLPT. This is repeated until the output of SigningKLPT is cyclic. Once this output is cyclic, the (compressed) isogeny $\mathbf{zip}$ corresponding to J is computed.

The final signature is then output as $(\mathbf{zip}, r, \mathbf{s})$, together with a boolean indicating whether or not the signing procedure succeeded.

3.5. Verification

The final algorithm to describe is the verification algorithm, which takes in the message, signature and public key of the signer, and checks whether the signature is valid. The verification consists of recomputing the response isogeny $\varphi_{\text{resp}} : E_A \to E_2$, the (dual of the) challenge isogeny $\widehat{\varphi_{\text{chall}}} : E_2 \to E_1$, before checking that the hash of E_1 and the message indeed generates the challenge isogeny.

Algorithm 26 SQIsign.Sign(sk, msg)

Input: Secret signing key sk and message msg $\in \{0,1\}^*$
Output: Signature σ
Output: found a boolean indicating whether computation succeeded

Parse sk as $(\alpha, B_{A,T}, Q)$
Compute $n(\alpha) := 2^e D_{\text{secret}}$
Compute $I_{\text{secret}} := \mathcal{O}_0\langle \bar{\alpha}, D_{\text{secret}}\rangle$
Compute $J_{\text{secret}} := \mathcal{O}_0\langle \alpha, 2^e\rangle$

Commitment.

Sample a, b at random from interval $[1, D_{\text{com}}]$ until $\gcd(a, b, D_{\text{com}}) = 1$
$I_{\text{com}} := \mathcal{O}_0\langle a + b\bar{\theta}, D_{\text{com}}\rangle$
$K_{\text{com}} := aP_0 + b\theta(P_0)$, where $P_0 \in B_{0,D_{\text{com}}}$
$E_1, \varphi_{\text{com}} := \mathsf{Normalized}(\mathsf{KernelToIsogeny}(K_{\text{com}}))$, where $\varphi_{\text{com}} : E_0 \to E_1$
$B_{1,D_{\text{chall}}} = \varphi_{\text{com}}(B_{0,D_{\text{chall}}})$. Write $B_{1,D_{\text{chall}}} = (P_1, Q_1)$

Challenge.

$K_{\text{chall}} := H(\text{msg}, E_1)$
$E_2, \varphi_{\text{chall}} := \mathsf{Normalized}(\mathsf{KernelToIsogeny}(K_{\text{chall}}))$, where $\varphi_{\text{chall}} : E_1 \to E_2$
$a, b := \mathsf{NormalizedDlog}_{D_{\text{chall}}}(E_1, (P_1, Q_1), K_{\text{chall}})$
$I_{\text{chall}} := [I_{\text{com}}]_* \mathsf{KernelDecomposedToIdeal}_{D_{\text{chall}}}(a, b)$
$(K, P) := \mathsf{CompleteBasis}_{D_{\text{chall}}, p+1}(E_1, K_{\text{chall}})$
$P' := \varphi_{\text{chall}}(P)$
$(P_2, Q_2) := \mathsf{TorsionBasis}_{D_{\text{chall}}, p+1}(E_2)$
$b_1 := 0,\ b_2 := 0$
$a_0, a_1 := \mathsf{NormalizedDlog}_{2^f}(E_2, ([3^g]P_2, [3^g]Q_2), [3^g]P')$
If $a_0 \equiv 0 \pmod 2$, swap a_0 and a_1 and set $b_1 := 1$
$s_1 := a_0^{-1} a_1 \pmod{2^f}$
$a_0, a_1 := \mathsf{NormalizedDlog}_{3^g}(E_2, ([2^f]P_2, [2^f]Q_2), [2^f]P')$
If $a_0 \equiv 0 \pmod 3$, swap a_0 and a_1 and set $b_2 := 1$
$s_2 := a_0^{-1} a_1 \pmod{3^g}$
$E_1', \widehat{\varphi_{\text{chall}}} := \mathsf{Normalized}(\mathsf{KernelToIsogeny}(P'))$, where $E_1' = E_1$
$\mathbf{s} := (b_1, s_1, b_2, s_2)$
Use $\mathbf{s}$ to compute deterministically a point Q' so that $\langle P', Q'\rangle = E_2[D_{\text{chall}}]$
$Q := \widehat{\varphi_{\text{chall}}}(Q')$
Compute r such that $K_{\text{chall}} = [r]Q$

Response.

$K := \overline{I_{\text{secret}}} \cdot I_{\text{com}} \cdot I_{\text{chall}}$
$e :=$ KLPT_signing_klpt_length
found := False
counter := 0
while not found and counter < SQIsign_response_attempts **do**
  counter := counter + 1
  J, found := $\mathsf{SigningKLPT}_{2^e}(K, I_{\text{secret}})$
  if not found **then**
    Continue
  end if
  Compute α such that $J \cdot \overline{I_{\text{chall}}} = \langle \alpha, 2^e D_{\text{chall}}\rangle$
  if If $\alpha/k \in \mathcal{O}_R(I_{\text{secret}})$ for any $k > 1$ **then**
    Continue
  end if
end while
_, **zip**, found := $\mathsf{IdealToIsogenyEichler}_{2^\bullet}(J, J_{\text{secret}}, B_{A,T}, Q)$
$\sigma := (\mathbf{zip}, r, \mathbf{s})$
return σ, found

Algorithm 27 $\mathsf{SQIsign.Verify}(\mathsf{msg}, \sigma, \mathsf{pk})$

Input: A message msg, signature σ and pk the public key
Output: verified a boolean indicating whether the verification passed
1: $\mathbf{zip}, r, \mathbf{s} := \sigma$
2: $E_2, Q_2 := \mathsf{Decompress}_{\mathsf{resp}}(\mathbf{zip}, \mathsf{pk})$
3: verified $:= \mathsf{DecompressAndCheck}_{\mathsf{chall}}(\mathbf{s}, E_2, Q_2, r, \mathsf{msg})$
4: **return** verified

On compression/decompression. Both the response isogeny and the dual of the challenge isogeny are compressed during signature generation, and made part of the signature. Both will be decompressed during verification. For the response isogeny, compression is given by factoring the isogeny into steps of length 2^f, then expressing each factor as an isogeny generated by $P + [s_i]Q$ for some deterministically generated basis P, Q of $E[2^f]$. In general, this requires potentially swapping P and Q; this is potentially done in the first step, indicated by the bit b. After the first step, we can take Q to be an element generating the dual of the previous step. Since the isogeny is not backtracking, this is sufficient to guarantee that the next kernel can be generated by an element of the form $P + [s_i]Q$. Additionally, the decompression, must return a point Q_2 of order 2 generating the dual of the last 2-isogeny computed. This is used to verify that the composition $\widehat{\varphi_{\text{chall}}} \circ \varphi_{\text{resp}}$ is cyclic.

The dual of the challenge isogeny consists only of a single step, of degree D_{chall}. Since D_{chall} is not a prime power, there is no guarantee that a point can be written as $P + [s_i]Q$ or $Q + [s_i]P$, where P, Q denote generators of $E[D_{\text{chall}}]$. Hence, we factor D_{chall} up into its prime power parts (in practice, a power of 2 and a power 3), and proceed as before. Finally, to check that the signature is valid, we check that the composition is cyclic, by checking that $\varphi_{\hat{\text{chall}}}$ does not factor through the isogeny generated by Q_2, before checking that the hash of E_1 and the message generates the challenge isogeny.

Algorithm 28 $\mathsf{Decompress}_{\mathsf{resp}}(E, s)$

Input: E a normalized Montgomery curve
Input: $\mathbf{s} = (b_1, s_1, s_2, \ldots, s_e)$ a compression of the isogeny φ
Output: E the codomain of φ (where E is a normalized Montgomery curve)
Output: $Q \in E$ a point generating the 2-isogeny φ' such that $\varphi = \widehat{\varphi'} \circ \varphi''$
1: $(P_{2^f}, Q_{2^f}) := \mathsf{TorsionBasis}_{2^f, p+1}(E)$
2: $b, \mathbf{s}' := \mathbf{s}$
3: **if** $b_1 = 1$ **then**
4: swap P_{2^f}, Q_{2^f}
5: **end if**
6: **for** s_i in $\mathbf{s}'$ **do**
7: $K := P_{2^f} + [s_i]Q_{2^f}$
8: $\varphi := \mathsf{KernelToIsogeny}(K)$
9: $E := \varphi(E)$
10: $Q_{2^f} := \varphi(Q_{2^f})$
11: $(Q, P_{2^f}) := \mathsf{CompleteBasis}_{2^f, p+1}(E, Q_{2^f})$
12: **end for**
13: $E', \psi := \mathsf{MontgomeryNormalize}(E)$
14: $Q := (0 : 0 : 1) \in E$
15: **return** E', $\psi(Q)$

3.6. Parameter sets

In this section, we list parameter sets for the security levels NIST-I, NIST-III and NIST-V. A parameter set consists in a choice of prime p (the characteristic of the field), and a bound B on the prime factors of T (so T is defined as the

Algorithm 29 $\mathsf{DecompressAndCheck}_{\mathsf{chall}}(E, s, Q, r, \mathsf{msg})$

Input: E a normalized Montgomery curve
Input: $\mathbf{s} = (b_1, s_1, b_2, s_2)$ a compression of the isogeny φ
Input: $Q \in E$ a point of order 2
Output: `verified` a boolean indicating whether verification should pass

$(P_{D_{\text{chall}}}, Q_{D_{\text{chall}}}) := \mathsf{TorsionBasis}_{D_{\text{chall}}}(E)$
$P_2, Q_2 := [3^g]P_{D_{\text{chall}}}, [3^g]Q_{D_{\text{chall}}}$
$P_3, Q_3 := [2^f]P_{D_{\text{chall}}}, [2^f]Q_{D_{\text{chall}}}$
if $b_1 = 1$ **then**
    swap P_2, Q_2
end if
if $b_2 = 1$ **then**
    swap P_3, Q_3
end if
$K_2 := P_2 + [s_1]Q_2$
if $[2^{f-1}]K_2 = Q$ **then**
    return FALSE
end if
$K_3 := P_3 + [s_2]Q_3$
$\varphi_2 := \mathsf{KernelToIsogeny}(K_2)$
$K_3 := \varphi_2(K_3)$
$\varphi_3 := \mathsf{KernelToIsogeny}(K_3)$
$\varphi := \varphi_3 \circ \varphi_2$
$E, \varphi := \mathsf{Normalized}(\varphi)$
Use s to compute deterministically Q' such that $\langle \ker \varphi, Q' \rangle = E[D_{\text{chall}}]$
$P := H(\mathsf{msg}, E)$
return $P = [r]\varphi(Q')$

largest odd integer dividing $p^2 - 1$ whose prime factors are at most B). All other parameters can be deduced from p, B, and the above specification; for convenience, we also list below the values of f and the factorisations of T.

Chapter 5 gives more details on parameter requirements, searches, and choices.

NIST-I:

$$\begin{aligned} p^{\text{I}}_{1973} &= \texttt{0x34e29e286b95d98c33a6a86587407437252c9e49355147ffffffffffffffff} \\ B^{\text{I}} &= 2000 \\ f &= 75 \\ T &= 3^{36} \cdot 7^4 \cdot 11 \cdot 13 \cdot 23^2 \cdot 37 \cdot 59^2 \cdot 89 \cdot 97 \cdot 101^2 \cdot 107 \cdot 109^2 \cdot 131 \cdot 137 \cdot 197^2 \cdot 223 \cdot 239 \\ &\quad \cdot 383 \cdot 389 \cdot 491^2 \cdot 499 \cdot 607 \cdot 743^2 \cdot 1033 \cdot 1049 \cdot 1193 \cdot 1913^2 \cdot 1973 \end{aligned}$$

NIST-III:

$$\begin{aligned} p^{\text{III}}_{47441} &= \texttt{0x3df6eeeab0871a2c6ae604a45d10ad665bc2e0a90aeb751c722f669356ea468} \\ &\quad \texttt{4c6174c1fffffffffffffffffffffffff} \\ B^{\text{III}} &= 48000 \\ f &= 97 \\ T &= 3^{68} \cdot 5 \cdot 7^{12} \cdot 11^4 \cdot 13 \cdot 47^4 \cdot 89 \cdot 113 \cdot 157^4 \cdot 173 \cdot 233 \cdot 239 \cdot 241 \cdot 443 \cdot 509^4 \cdot 569 \cdot 761^4 \\ &\quad \cdot 1229 \cdot 2393 \cdot 3371 \cdot 4517 \cdot 5147 \cdot 5693 \cdot 5813 \cdot 9397 \cdot 26777 \cdot 39679 \cdot 47441 \end{aligned}$$

NIST-V:

$$p^{\mathrm{V}}_{318233} = \texttt{0x255946a8869bc68c15b0036936e79202bdbe6326507d01fe3ac5904a0dea65fa}$$
$$\texttt{f0a29a781974ce994c68ada6e1fffffffffffffffffffffffffffffff}$$
$$B^{\mathrm{V}} = 320000$$
$$f = 145$$
$$T = 3^{72} \cdot 5 \cdot 7 \cdot 13^6 \cdot 17 \cdot 37 \cdot 41^6 \cdot 53 \cdot 67^6 \cdot 73 \cdot 103^6 \cdot 127 \cdot 151 \cdot 461^6 \cdot 643 \cdot 733 \cdot 739 \cdot 827^6 \cdot 1009$$
$$\cdot 2539 \cdot 4153 \cdot 5059 \cdot 7127 \cdot 10597 \cdot 13591 \cdot 14923 \cdot 15541 \cdot 15991 \cdot 18583 \cdot 23227 \cdot 48187$$
$$\cdot 63247 \cdot 65521 \cdot 318233$$

3.7. Binary format

For the purpose of transmitting the mathematical objects involved in the signature scheme over the wire, we have to specify how they are encoded into bytes. The following types of component objects are involved:

- Elements of $\mathbb{F}_p$ are encoded as unsigned integers between 0 and $p-1$, in little-endian, using the smallest number of bytes capable of representing all elements of $\mathbb{F}_p$, i.e., $\lceil \log_{256}(p) \rceil$ bytes.
- Elements of $\mathbb{F}_{p^2}$ are encoded by simply concatenating the encoding of the real part with the encoding of the imaginary part.
- Integers in $\mathbb{Z}$ are encoded in little-endian two's complement representation; the number of bytes is fixed on a per-instance basis by the specification.
- Quaternions are encoded by storing the encodings of five integers: The denominator, followed by four coefficients for the basis $1, i, j, k$.
- Elliptic-curve points are encoded as the encoding of the x-coordinate. Torsion bases (P, Q) are encoded by concatenating the encodings of $P, Q, P-Q$.

Secret keys. The secret key is encoded by concatenating the encodings of the following objects, in order:

- the associated public key;
- the quaternion α defined in Algorithm 25;
- a point generating the kernel of the dual of φ_{secret};
- the "curve" part B_{A,T^+} of the torsion basis $B_{A,T}$;
- the "twist" part B_{A,T^-} of the torsion basis $B_{A,T}$.

Public keys. The public key is encoded as a single element of $\mathbb{F}_{p^2}$, the A-coefficient of the normalized Montgomery curve in the isomorphism class of the public-key curve.

Signatures. The signature is encoded by concatenating the encodings of the following objects, in order:

- the compressed form $\mathbf{zip}$ of the isogeny φ_I, encoded as a single byte with value b, followed by the integers $s_1, ..., s_g$ represented using $\lceil f/8 \rceil$ bytes each, notation from Algorithm 19;
- the integer r from Step 28 of Algorithm 26;
- the tuple $\mathbf{s}$ from Step 25 of Algorithm 26, encoded as a single byte with value $(b_1 + 2b_2)$, followed by the encodings of the integers s_1 and s_2 using the minimal number of bytes required to represent values in $\{0, ..., D_{\text{chall, even}} - 1\}$ resp. $\{0, ..., D_{\text{chall, odd}} - 1\}$.

CHAPTER 4

Known answer test values

The folder **KAT** of the submission media file contains KAT files for SQIsign parameters of level NIST-I, NIST-III and NIST-V. Each KAT file contains values for secret keys (sk), public keys (pk), signatures concatenated with messages (sm) and seeds (seed).

CHAPTER 5

Parameter search and choices

This chapter describes parameter requirements, searches, and choices for the instantiations of SQIsign in the NIST-I, NIST-III, and NIST-V security levels. We emphasize that the specific choice of parameters within the requirements below has no impact on the security of SQIsign (see Chapter 9).

5.1. Parameter requirements

The most expensive computational task in SQIsign is the computation of isogenies. We work with supersingular elliptic curves over $\mathbb{F}_{p^2}$ for a prime p with cardinality $(p+1)^2$, and their twists of cardinality $(p-1)^2$. As described in Chapter 2, we can compute isogenies of any smooth degree $N \mid (p^2-1)$ efficiently without moving to field extensions of $\mathbb{F}_{p^2}$. Hence, the efficiency of SQIsign hinges on choosing a prime p of suitable size, such that (p^2-1) contains sufficiently many smooth prime factors.

As detailed in Chapter 9, the following sizes are prescribed for security parameter λ:

- Prime sizes of $\log_2(p) \approx 2\lambda$.
- The degree D_{com} of the commitment isogeny φ_{com} of size approximately $2^{2\lambda} \approx p$.
- The degree D_{chall} of the challenge isogeny φ_{chall} of size approximately $2^{\lambda} \approx p^{1/2}$.
- The degree ℓ^e of the signature isogeny φ_{resp} of size approximately $p^{15/4}$.
- The degree T of isogenies used in Algorithm 19 of size approximately $p^{5/4}$.

Our parameter choices mainly prioritize fast verification, while maintaining reasonable signing performance. Since the verifier computes the isogenies φ_{resp} and $\widehat{\varphi_{\text{chall}}}$, we choose the respective degrees to be as smooth as possible. In particular, we fix $\ell = 2$, such that φ_{resp} is of degree 2^e. Since T has to be coprime to the degree of φ_{resp}, this yields the requirement $T' = 2^f \cdot T \mid (p^2-1)$. We present our more specific requirements below, and refer to Section 3.6 for explicit choices.

(1) Optimized implementations of finite field arithmetic benefit from at least two bits of distance to the next multiple of the wordlength of 64 bits. Hence, we aim for primes p of at most 254, 382, resp. 510 bits for NIST security levels I, III, resp. V.

(2) For efficient computation of T-isogenies, we require the odd factor $T \approx p^{5/4}$ to be B-smooth for B as small as possible. Since values of T close to $p^{5/4}$ may lead to frequent failures, we require $T > p^{1.27}$ for security levels I and III, resp. $T > p^{1.26}$ for security level V as a failure margin.

(3) We restrict to primes $p \equiv 3 \pmod 4$, since this less general case significantly simplifies implementations and is beneficial for fast field arithmetic over $\mathbb{F}_{p^2}$.

(4) The signature isogeny φ_{resp} of degree 2^e is computed as a composition of $\lceil e/f \rceil$ isogenies of degree 2^f, where 2^f is the maximal power of two dividing $p+1$. Since the bottleneck is the generation of kernel points for each of the 2^f-isogenies, we require f to be as large as possible, in order to reduce the number of kernel point generations.

(5) To optimize the cost for computing the challenge isogeny φ_{chall}, we set $D_{\text{chall}} = 2^f \cdot 3^g$ of size roughly $p^{1/2}$, and require $D_{\text{chall}} \mid (p^2-1)$.

(6) To simplify the kernel point generation during the recomputation of the challenge isogeny, we additionally require that $D_{\text{chall}} | (p+1)$. We note that allowing $3^g \mid (p-1)$ instead would not have a large impact on performance, but slightly complicate the implementation.

These requirements induce a certain conflict between requirements (2) and (4): Larger values of f leave a smaller factor of p^2-1 (of $\log_2(2p-f-1)$ bits), from which we can pick smooth factors for T. As mentioned above, we aim at fast verification, thus prioritizing (4) over (2).

Cost metric. For the initial assessment of parameters, we use a cost metric. For signing, the main bottleneck is the repeated computation of T-isogenies, where the number of computation depends on the size of f. Hence, for the prime factorization $T = \prod \ell_i^{e_i}$ we estimate the cost of a T-isogeny through

$$C_{\text{isog}}(T) = \sum e_i \cdot \min\{C_{\text{Vélu}}(\ell_i), C_{\sqrt{\text{élu}}}(\ell_i)\},$$

where we use the cost estimates of [ACR23] for optimized Vélu-, resp. $\sqrt{\text{élu}}$-isogenies as

$$C_{\text{Vélu}}(\ell) = 6\ell \quad \text{and} \quad C_{\sqrt{\text{élu}}}(\ell) = 37m^{log_2(3)} + 5m\log_2(m) + 6\log_2(m) + \frac{42}{3}m + \frac{77}{6}$$

for $m = \lfloor\sqrt{\ell-1}/2\rfloor$. To account for the number of repetitions of T-isogenies during signing, we use the signing cost metric

$$C_S(p) = \frac{C_{\text{isog}}(T)}{f}.$$

This represents a more fine-grained metric compared to the metric $\sqrt{B}/f$ from [DLLW23].

The performance estimate for verification only considers the size of f.

5.2. Finding parameters

In this section we discuss the different approaches to searching for suitable parameters. We note that other methods from the literature find somewhat SQIsign-friendly primes, but fail to satisfy all of our requirements simultaneously. In particular, the methods from [Cos20, DKL+20, DLLW23] find parameters with reasonable powers of two and three, and smoothness bounds B, but fail to satisfy $(2^f \cdot 3^g) \mid (p+1)$. The method from [CMN21] finds smooth parameters, but does not provide large powers of two and three. The method from [BSC+22] finds smooth parameters and large powers of two, but does not provide large powers of two and three together.

5.2.1. Sieve-and-boost

We search for primes of the form

$$p = 2(2^{f'}3^{g'}x)^n - 1$$

for a smooth number x, and for different values of f', g' such that $2^{nf'} \cdot 3^{ng'} \geq \sqrt{p}$, to ensure that φ_{chall} can be computed as a chain of 2-isogenies and 3-isogenies. This idea is adapted from [Cos20], which is based on the observation that when $p_n(x) := 2x^n - 1$, then

$$p_n(x)^2 - 1 = 4x^n \prod_{d|n} \Phi_d(x),$$

where Φ_d denotes the d-th cyclotomic polynomial (see [BSC+22]).

When searching for primes of bitsize smaller than $256, 384$ and 512 bits, respectively, the search space for x is roughly of size $\log_2(p)/2n$ bits. For suitable values of n, we can exhaust the full search space of potential smooth values of x, but in other cases, we artificially reduce the size of x by increasing the size of $\log_2(2^{f'} \cdot 3^{g'})$.

The algorithm starts by identifying smooth numbers in the search interval for x, using the sieve implementation from [CMN21]. For each smooth x and suitable values of f', g', we store the primes of the form $p(x) = 2(2^{f'}3^{g'}x)^{n_{\text{lvl}}} - 1$, before simultaneously computing the B-smooth parts of p^2-1 by using a product tree [Ber04]. The numbers $n_{\text{lvl}} \in \{n_{\text{I}}, n_{\text{III}}, n_{\text{V}}\}$ are chosen to give primes suitable for the three security levels.

Note that the balance between the sizes of f' and g' represents the balance between requirement (2) and (4) as discussed above.

For an overview over what ranges we sieved over, and values of n, see Table 1. The probabilities of finding large enough smooth T are shown in Table 2. Note that for large values of n, the search space is not sufficient to expect to

Sieve range	n_{I} for NIST-I	n_{III} for NIST-III	n_{V} for NIST-V
$[2^{45}, 2^{46}]$	-	4	-
$[2^{41}, 2^{42}]$	3	-	6
$[2^{31}, 2^{32}]$	4	6	8
$[2^{20}, 2^{21}]$	6	9	12
$[2^{15}, 2^{16}]$	8	12	16
$[2^{9}, 2^{10}]$	12	18	24

TABLE 1. List of ranges that have been exhausted for parameters using the sieve-and-boost approach.

	n	$\log_2(r)$	**Probability of** $p_n(r)^2 - 1$ **having odd** B**-smooth** $T \approx p_n^{5/4}$ **given** $r = 2^{f'}3^{g'}x$
NIST-I	2	127.0	$\approx 2^{-34.1}$
$B = 2^{11}$	3	84.7	$\approx 2^{-35.3}$
$\log_2(p) = 254$	4	63.5	$\approx 2^{-28.7}$
$f = 64$	6	42.3	$\approx 2^{-24.4}$
	8	31.8	$\approx 2^{-26.7}$
NIST-III	2	191.0	$\approx 2^{-39.4}$
$B = 2^{15}$	3	127.3	$\approx 2^{-40.7}$
$\log_2(p) = 382$	4	95.5	$\approx 2^{-33.4}$
$f = 96$	6	63.7	$\approx 2^{-28.6}$
	8	47.8	$\approx 2^{-31.1}$
NIST-V	2	255.0	$\approx 2^{-46.1}$
$B = 2^{18}$	3	170.0	$\approx 2^{-47.5}$
$\log_2(p) = 510$	4	127.5	$\approx 2^{-39.2}$
$f = 128$	6	85.0	$\approx 2^{-33.7}$
	8	63.8	$\approx 2^{-36.5}$

TABLE 2. Assuming that $r = 2^{f'}3^{g'}x$ where x is B-smooth and p has $\log_2(p)$ bits, we display the probability of $p = p_n(r)^2 - 1$ having a B-smooth divisor $2^f T \mid (p^2 - 1)$ with $T \approx p^{5/4}$.

find SQISIGN-friendly primes. Especially for the NIST-I security level, the success probabilities of this approach are limited.

We ran this approach with all choices of $f' \in \{\lfloor 50/n_{\mathrm{I}} \rfloor, \ldots, \lfloor 100/n_{\mathrm{I}} \rfloor\}$, and for each such f', setting g' to be the an appropriate size satisfying $2^{nf'+1} \cdot 3^{ng'} \geq \sqrt{p}$. We sieved for 2^{10}-smooth numbers as inputs x. The smoothness bound B of T was chosen per value of f' and n_{lvl}, such that $\sqrt{B}/f' < \texttt{cost\_bound}_{\texttt{lvl}}$ for $\texttt{cost\_bound}_{\texttt{lvl}} \in \{1, 2.5, 5\}$ corresponding to the security levels, estimating signing cost in the rough metric $\sqrt{B}/f$ from [**DLLW23**].

5.2.2. XGCD-and-boost

For the smaller choices of n, where we cannot exhaust the full search interval, we use the technique of [**BSC+22**] to improve the probabilities of finding suitable parameters by first generating pairs $(r, r-1)$ of smooth numbers and using them as inputs to $p(x)$. Since both x^n and $x - 1$ are divisors of $p_n(x)^2 - 1$, we are guaranteed a smooth factor T' of $\frac{n+1}{n}(\log_2(p) - 1) + 2$ bits in this case.

We use this technique specifically for NIST-I parameters and $n = 2$, requiring roughly 128-bit $(r, r-1)$. This guarantees a smooth factor $T' \approx p^{3/2}$, which is sufficient when $f \approx \log_2(p)/4$.

For finding suitable inputs, we adapt the techniques of [Cos20, DKL+20] using the extended Euclidean algorithm (XGCD). We fix smooth coprime integers a, b such that $ab \approx 2^{128}$. Then the XGCD allows us to find $s, t \in \mathbb{Z}$ with $st \approx ab$ such that $as + bt = 1$. We pick $(r, r-1) = (|as|, |bt|)$, resp. $(r, r+1) = (|as|, |bt|)$, which are B-smooth if and only if s and t are B-smooth. The probability for this can be determined through the Dickman-De Bruijn function (see [CMN21]). In our context, this is maximized by choosing $a \approx b \approx 2^{64}$, such that $s \approx t \approx 2^{64}$, and is roughly $2^{-29.7}$ for $B = 2^{11}$, resp. $2^{-25.6}$ for $B = 2^{12}$.

In order to satisfy the SQIsign requirements, we pick $a = 2^{f'} \cdot 3^{g'}$, such that potential primes $p = p_2(r)$ satisfy $a^2 = 2^{2f'} \cdot 3^{2g'} \mid (p+1)$ and $f \geq 2f' + 1$. For b, we sample six (not necessarily distinct) 2^{10}-smooth primes ℓ_i, and find suitable powers of 5 and 7, such that $b = 5^k \prod_{i=1}^{6} \ell_i \approx 2^{64}$, resp. $b = 7^k \prod_{i=1}^{6} \ell_i \approx 2^{64}$. This results in a search space of roughly $2^{36.1}$ per choice of (f', g').

Furthermore, we can increase the search space by iterating through multiple XGCD solutions of appropriate size (see [Cos20]). Given the solution (s, t), there is an infinite number of solutions of the form $(s_j, t_j) = (s + jb, t - ja)$ with $j \in \mathbb{Z}$. However, only a limited number of these solution is suitable for our purposes due to the size requirement of $r = |as_j| < 2^{128}$. On average, this increases the search space by factor 4 in our configuration. Given the success probabilities above, we thus expect to find enough suitable pairs $(r, r \pm 1)$ for $B = 2^{11}$.

Concretely, our implementation does not require $(r, r \pm 1)$ to be fully smooth. Instead, for reasons of efficiency, we first check if $p_2(r)$ is prime. If it is, we check $p_2(r)^2 - 1$ for a large enough odd smooth factor T. This allows for $r \pm 1$ to have non-smooth factors, in case $r \mp 1$ contains enough smooth factors for T to be large enough. Thus, this increases the chances for finding suitable parameters.

Furthermore, a variant of this approach first samples one resp. two small primes factors ℓ_1 (and ℓ_2), and sets $a = 2^{f'} \cdot 3^{g'} \cdot \ell_1$ resp. $a = 2^{f'} \cdot 3^{g'} \cdot \ell_1 \cdot \ell_2$. In contrast, we sample only five resp. four small primes factors and compute b as above, such that $a \approx b \approx 2^{64}$. Since $r = |as| \approx 2^{128}$, this reduces the size of s, and thus increases the smoothness probability for r. This is beneficial, since we have $r^2 \mid (p_2(r)^2 - 1)$, while the factor $r \mp 1$ can compensate for the worse smoothness probabilities of $r \pm 1$.

We ran this approach with various choices of $f' \in [32, 40]$ and appropriate $g' \in [15, 20]$, such that $2^{2f'} \cdot 3^{2g'} \approx 2^{64}$. The smoothness bound B was chosen as $B = 2^{11}$ for $f' \leq 35$, and $B = 2^{12}$ for $f' \geq 36$, representing the tradeoff between smoothness of T and size of f. Due to the sampling of small primes, this approach is probabilistic. Given the relatively small search spaces and the number of trials performed, we expect to have covered the full search space for the given parameters.

5.2.3. Results

We collected a list of primes found by the described searches. In a postprocessing step, we assembled a shortlist of the best potential parameters per security level in the following logic. The shortlist starts from the prime with the largest f and corresponding smallest signing cost metric $C_S(p)$. We then go through all sizes of f in descending order, adding at most one prime per f, for which $C_S(p)$ is smaller than for any other prime on the shortlist, and the smallest among all primes featuring this f. We give the best primes from the resulting shortlists in reverse order, displaying primes in hexadecimal form.

As expected from Table 2, the sieve-and-boost method performed especially well for NIST-III and NIST-V, using $n_{\text{III}} \in \{4, 6\}$,[1] resp. $n_{\text{V}} = 6$. For NIST-I, XGCD-and-boost is superior, as discussed above.

As shown in Section 3.6, we chose the primes $p^{\text{I}}_{1973}, p^{\text{III}}_{47441}$, resp. p^{V}_{318233} for our instantiations for security levels I, III, resp. V. These choices feature large enough f for fast verification, with small enough smoothness bounds B to keep signing reasonably efficient.

We note that since all values of f in our instantiations satisfy $2^f > p^{1/4}$, our implementations do not consider various special cases that may arise for values of f such that $2^f < p^{1/4}$.

[1] Note that $n_{\text{III}} = 6$ leads to a relatively small search space.

5.2.3.1. NIST-I.

- $p^{\mathrm{I}}_{1223} = \texttt{0xea6a4dda9518e5c5d50ccdfbd97e4c49efe85e0e09039c7fffffffffffffff}$
  $\log_2(p^{\mathrm{I}}_{1223}) \approx 251.9$
  $f = 67$
  $T = 3^{38} \cdot 5^3 \cdot 7^6 \cdot 11^4 \cdot 13 \cdot 19 \cdot 31 \cdot 47 \cdot 59^2 \cdot 89 \cdot 137^2 \cdot 139 \cdot 173 \cdot 197 \cdot 257^2 \cdot 281 \cdot 311 \cdot 347^2 \cdot 353 \cdot 359 \cdot 439^4 \cdot 577 \cdot 733 \cdot 853 \cdot 983 \cdot 1223$
  $\log_2(T)/\log_2(p^{\mathrm{I}}_{1223}) \approx 1.274$
  $C_S(p^{\mathrm{I}}_{1223}) \approx 594.6$
  Method: XGCD-and-boost ($f' = 33$, $g' = 19$, $B = 2^{11}$, one extra factor ℓ_i in a, powers of 5 in b)

- $p^{\mathrm{I}}_{1973} = \texttt{0x34e29e286b95d98c33a6a86587407437252c9e49355147ffffffffffffffff}$
  $\log_2(p^{\mathrm{I}}_{1973}) \approx 253.7$
  $f = 75$
  $T = 3^{36} \cdot 7^4 \cdot 11 \cdot 13 \cdot 23^2 \cdot 37 \cdot 59^2 \cdot 89 \cdot 97 \cdot 101^2 \cdot 107 \cdot 109^2 \cdot 131 \cdot 137 \cdot 197^2 \cdot 223 \cdot 239 \cdot 383 \cdot 389 \cdot 491^2 \cdot 499 \cdot 607 \cdot 743^2 \cdot 1033 \cdot 1049 \cdot 1193 \cdot 1913^2 \cdot 1973$
  $\log_2(T)/\log_2(p^{\mathrm{I}}_{1973}) \approx 1.295$
  $C_S(p^{\mathrm{I}}_{1973}) \approx 743.8$
  Method: XGCD-and-boost ($f' = 35$, $g' = 18$, $B = 2^{11}$, two extra factors ℓ_i in a, powers of 7 in b)

- $p^{\mathrm{I}}_{8011} = \texttt{0x31ebc32c245c72c40115748f25c4ba516cb58aaae247ffffffffffffffffffff}$
  $\log_2(p^{\mathrm{I}}_{8011}) \approx 253.6$
  $f = 83$
  $T = 3^{30} \cdot 5^2 \cdot 7^7 \cdot 17^2 \cdot 31 \cdot 61 \cdot 71^2 \cdot 173^2 \cdot 241 \cdot 349^2 \cdot 409 \cdot 521 \cdot 577^2 \cdot 653^2 \cdot 761 \cdot 827^2 \cdot 1283 \cdot 2011 \cdot 2017 \cdot 2521^2 \cdot 3181 \cdot 3833 \cdot 3931 \cdot 8011$
  $\log_2(T)/\log_2(p^{\mathrm{I}}_{8011}) \approx 1.301$
  $C_S(p^{\mathrm{I}}_{8011}) \approx 1247.1$
  Method: XGCD-and-boost ($f' = 40$, $g' = 15$, $B = 2^{12}$, two extra factors ℓ_i in a, powers of 7 in b)

5.2.3.2. NIST-III.

- $p^{\mathrm{III}}_{5563} = \texttt{0x4cd95e35908847e31ac2953eb6d35610ccd37a339b81a09214ad43375dd1219f9ed34f2a4ad05b5507fffffffffffff}$
  $\log_2(p^{\mathrm{III}}_{5563}) \approx 378.3$
  $f = 55$
  $T = 3^{84} \cdot 5 \cdot 7 \cdot 13^{12} \cdot 19 \cdot 29 \cdot 43 \cdot 47^6 \cdot 79 \cdot 109 \cdot 229 \cdot 277 \cdot 433 \cdot 457^6 \cdot 463 \cdot 719 \cdot 757 \cdot 953^6 \cdot 1321 \cdot 1399 \cdot 1627 \cdot 2179 \cdot 2293 \cdot 4357 \cdot 5563$
  $\log_2(T)/\log_2(p^{\mathrm{III}}_{5563}) \approx 1.287$
  $C_S(p^{\mathrm{III}}_{5563}) \approx 1752.1$
  Method: Sieve-and-boost using $n = 6$

- $p^{\mathrm{III}}_{22741} = \texttt{0x851b4a8ba9ca5268304fcfea6b20d3641c5982a3e888543d00f3741c8764bdbef38bf6a1531aa1ffffffffffffffff}$
  $\log_2(p^{\mathrm{III}}_{22741}) \approx 379.1$
  $f = 69$
  $T = 3^{84} \cdot 5 \cdot 7 \cdot 11 \cdot 19 \cdot 29^4 \cdot 31^4 \cdot 47 \cdot 71 \cdot 113 \cdot 163 \cdot 191 \cdot 193 \cdot 229^4 \cdot 293^4 \cdot 373 \cdot 409 \cdot 463 \cdot 571^4 \cdot 599^4 \cdot 757 \cdot 881 \cdot 1033 \cdot 1489 \cdot 1753 \cdot 8069 \cdot 10831 \cdot 22741$
  $\log_2(T)/\log_2(p^{\mathrm{III}}_{22741}) \approx 1.270$
  $C_S(p^{\mathrm{III}}_{22741}) \approx 1939.7$
  Method: Sieve-and-boost using $n = 4$

- $p^{\mathrm{III}}_{47441} = \texttt{0x3df6eeeab0871a2c6ae604a45d10ad665bc2e0a90aeb751c722f669356ea4684c6174c1fffffffffffffffffffffff}$
  $\log_2(p^{\mathrm{III}}_{47441}) \approx 377.9$

$f = 97$
$T = 3^{68} \cdot 5 \cdot 7^{12} \cdot 11^4 \cdot 13 \cdot 47^4 \cdot 89 \cdot 113 \cdot 157^4 \cdot 173 \cdot 233 \cdot 239 \cdot 241 \cdot 443 \cdot 509^4 \cdot 569 \cdot 761^4$
$\cdot 1229 \cdot 2393 \cdot 3371 \cdot 4517 \cdot 5147 \cdot 5693 \cdot 5813 \cdot 9397 \cdot 26777 \cdot 39679 \cdot 47441$
$\log_2(T)/\log_2(p^{\mathrm{III}}_{47441}) \approx 1.298$
$C_S(p^{\mathrm{III}}_{47441}) \approx 2987.3$
Method: Sieve-and-boost using $n = 4$

- $p^{\mathrm{III}}_{194581} =$ `0x2a61eff6f5b99e8a6531a3dd016bce053791af1d1f4c95da3643c770c28ca9e1fffffffffffffffffffffffffffffff`
$\log_2(p^{\mathrm{III}}_{194581}) \approx 381.4$
$f = 129$
$T = 3^{60} \cdot 5 \cdot 7 \cdot 13 \cdot 19^3 \cdot 29^4 \cdot 37^2 \cdot 41 \cdot 59^4 \cdot 89 \cdot 113 \cdot 139 \cdot 277 \cdot 353 \cdot 431^4 \cdot 541 \cdot 557 \cdot 743$
$\cdot 967^8 \cdot 3469 \cdot 4993 \cdot 6221 \cdot 6761 \cdot 8233 \cdot 9521 \cdot 25169 \cdot 35869 \cdot 54493 \cdot 194581$
$\log_2(T)/\log_2(p^{\mathrm{III}}_{194581}) \approx 1.293$
$C_S(p^{\mathrm{III}}_{194581}) \approx 3932.1$
Method: Sieve-and-boost using $n = 4$

5.2.3.3. NIST-V.

- $p^{\mathrm{V}}_{40609} =$ `0x1258a04d42f6b813d52a2b7a316c88f20e534878009f8262082fa9996b5bb08ed14526e626a06c28e0388a0721ebd5514e072a9d10861ffffffffffffffffff`
$\log_2(p^{\mathrm{V}}_{40609}) \approx 504.2$
$f = 73$
$T = 3^{114} \cdot 5 \cdot 7^6 \cdot 13 \cdot 17 \cdot 19 \cdot 23 \cdot 37 \cdot 43 \cdot 59^6 \cdot 97 \cdot 113^6 \cdot 211 \cdot 229^6 \cdot 263 \cdot 373^6 \cdot 487 \cdot 547 \cdot 929^6$
$\cdot 1021 \cdot 1297 \cdot 1511 \cdot 2053 \cdot 2593 \cdot 2647 \cdot 6217 \cdot 19777 \cdot 24103 \cdot 30403 \cdot 40609$
$\log_2(T)/\log_2(p^{\mathrm{V}}_{40609}) \approx 1.264$
$C_S(p^{\mathrm{V}}_{40609}) \approx 3804.4$
Method: Sieve-and-boost using $n = 6$

- $p^{\mathrm{V}}_{66343} =$ `0x353849c34fc94533c0d441876a7a45402ba8961efb78f3aef29a679fc7623bfcc519c82a014ac7fa64b2f97fa47d154b20f99af07ffffffffffffffffffffff`
$\log_2(p^{\mathrm{V}}_{66343}) \approx 505.7$
$f = 91$
$T = 3^{108} \cdot 5 \cdot 7^2 \cdot 13^6 \cdot 37^2 \cdot 43 \cdot 79 \cdot 107 \cdot 139 \cdot 211 \cdot 349 \cdot 421^6 \cdot 433^6 \cdot 439 \cdot 457 \cdot 467 \cdot 739^6$
$\cdot 947^6 \cdot 1433 \cdot 1511 \cdot 2287 \cdot 3229 \cdot 5153 \cdot 5399 \cdot 5953 \cdot 9133 \cdot 51637 \cdot 61441 \cdot 66343$
$\log_2(T)/\log_2(p^{\mathrm{V}}_{66343}) \approx 1.272$
$C_S(p^{\mathrm{V}}_{66343}) \approx 4282.2$
Method: Sieve-and-boost using $n = 6$

- $p^{\mathrm{V}}_{141079} =$ `0xf18c5c8a0bafce13183dc4b177b859181420c5d9aa73483b708189cf1f67db4aec4488346a2cfe7fc41c079c2123aea07fffffffffffffffffffffffffffff`
$\log_2(p^{\mathrm{V}}_{141079}) \approx 499.9$
$f = 115$
$T = 3^{90} \cdot 5^6 \cdot 7^{12} \cdot 13 \cdot 19 \cdot 23^{12} \cdot 31 \cdot 67 \cdot 89 \cdot 157 \cdot 197^6 \cdot 223^6 \cdot 251^6 \cdot 373 \cdot 433 \cdot 487 \cdot 571 \cdot 659$
$\cdot 757 \cdot 769 \cdot 983 \cdot 991 \cdot 4597 \cdot 4937 \cdot 5717 \cdot 21757 \cdot 28513 \cdot 30931 \cdot 73877 \cdot 78193 \cdot 141079$
$\log_2(T)/\log_2(p^{\mathrm{V}}_{141079}) \approx 1.264$
$C_S(p^{\mathrm{V}}_{141079}) \approx 4891.7$
Method: Sieve-and-boost using $n = 6$

- $p^{\mathrm{V}}_{318233} =$ `0x255946a8869bc68c15b0036936e79202bdbe6326507d01fe3ac5904a0dea65faf0a29a781974ce994c68ada6e1fffffffffffffffffffffffffffffffffffff`
$\log_2(p^{\mathrm{V}}_{318233}) \approx 501.2$
$f = 145$

$T = 3^{72} \cdot 5 \cdot 7 \cdot 13^6 \cdot 17 \cdot 37 \cdot 41^6 \cdot 53 \cdot 67^6 \cdot 73 \cdot 103^6 \cdot 127 \cdot 151 \cdot 461^6 \cdot 643 \cdot 733 \cdot 739 \cdot 827^6 \cdot 1009$
$\cdot 2539 \cdot 4153 \cdot 5059 \cdot 7127 \cdot 10597 \cdot 13591 \cdot 14923 \cdot 15541 \cdot 15991 \cdot 18583 \cdot 23227 \cdot 48187$
$\cdot 63247 \cdot 65521 \cdot 318233$
$\log_2(T)/\log_2(p^{\mathrm{V}}_{318233}) \approx 1.291$
$C_S(p^{\mathrm{V}}_{318233}) \approx 5764.2$
Method: Sieve-and-boost using $n = 6$

CHAPTER 6

Performance analysis

The submission package includes a reference implementation written in portable C (C99) and an additional, assembly-optimized implementation targeting the Intel Broadwell architecture. Both implementations share the same code base of the overall SQIsign library. The SQIsign library is built into several sub-modules that are linked to libraries supporting the NIST Signature API:

- Protocols: implementing the signature key-generation, signing and verification protocols.
- Id2iso: implementation of ideal to isogeny algorithms.
- EC: elliptic curve and isogeny computation.
- KLPT: implementation of the KLPT algorithm.
- Precomp: module to pre-compute constants for the code package.
- Quaternion: module for quaternion computation.
- Intbig: arbitrary precision module based on GMP.
- Common: common dependencies.

The SQIsign library contains a test harness for self-tests and KAT verification, and a benchmarking application reporting CPU cycles. Each submodule contains further unit-testing suites. All build and test options are described in the `README.md` file along with the submission package. While the implementation provides from scratch implementations of all SQIsign building blocks except Intbig, it is not constant-time.

6.1. Key and signature sizes

Key and signature sizes are listed in Table 1 for each security level.

Table 1. SQIsign key and signature sizes in bytes for each security level.

Parameter set	`Public key`	`Secret key`	`Signature`
NIST-I	64	782	177
NIST-III	96	1138	263
NIST-V	128	1509	335

6.2. Reference implementation

For benchmarking, the reference implementation is built with two configurations: (1) using the GMP system installation on Ubuntu 22.04 LTS and (2) using a custom built version with disabled assembly code. The CMake build options were:

(1) `-DSQISIGN_BUILD_TYPE=ref -DCMAKE_BUILD_TYPE=Release`
(2) `-DSQISIGN_BUILD_TYPE=ref -DCMAKE_BUILD_TYPE=Release -DENABLE_GMP_BUILD=ON -DGMP_BUILD_CONFIG_ARGS:STRING="--disable-assembly"`

6.3. Optimized implementation

The optimized implementation is the same as the reference implementation.

6.4. Intel Broadwell optimized implementation

The additional implementation uses assembly-optimized code targeting the Intel Broadwell architecture (or later) and is provided for NIST-I. The assembly-optimizations are applied to the GF-module, while the remaining code base is re-used from the reference implementation. For benchmarking, we used the following CMake options: `-DSQISIGN_BUILD_TYPE=broadwell -DCMAKE_BUILD_TYPE=Release`.

6.5. Performance evaluation

Performance evaluation was performed on an Intel x86 64 bit CPU. All builds use the compile flags `-march=native -O3`. Turbo Boost was turned off during benchmarking to get consistent timings. GMP was used in version 6.2.1.

The results are shown in Table 2.

TABLE 2. SQISIGN performance in 10^6 CPU cycles on an Intel Xeon Gold 6338 CPU (Ice Lake), compiled on Ubuntu with clang version 14. Results are the median of 10 benchmark runs.

Parameter set	`KeyGen`	`Sign`	`Verify`
Reference implementation (with default GMP installation)			
NIST-I	2'834	4'781	103
NIST-III	21'359	38'884	687
NIST-V	84'944	160'458	2'051
Reference implementation (with GMP `--disable-assembly`)			
NIST-I	3'728	5'779	108
NIST-III	23'734	43'760	654
NIST-V	91'049	158'544	2'177
Assembly-optimized implementation for Intel Broadwell or later			
NIST-I	1'661	2'370	37

CHAPTER 7

Implementation details

This section introduces some implementation details and constants that are used in the various algorithms of the protocols. We will explain their role and describe how they are computed. The value of some of those constants is chosen to ensure that some failure probability is smaller than some value that we define as negl. The heuristics behind the formula we used are the object of Chapter 8.

For the concrete computations, we take $\mathsf{negl} = 2^{-64}$.

7.1. Generic

Constants.

(1) `KLPT_primality_num_iter`: the number of Miller-Rabin tests used in to test pseudo-primality for the equivalent ideal functionality. It should be such that $4^{-\text{KLPT\_equiv\_primality\_num\_iter}} < \mathsf{negl}$ (this is a common upper-bound on the probability of failure of repetitions of Miller Rabin test).

(2) `KLPT_random_prime_attempts`: when looking for a random prime number of bitsize k, we try at most k(`KLPT_random_prime_attempts`) random integers until one that is prime is found. Should ensure that

$$(1 + \ln 2/k)^{k(\texttt{KLPT\_random\_prime\_attempts})} \leq \mathsf{negl}$$

This comes from the fact that a random number of bitsize k has a probability $-\ln 2/k$ of being a prime.

7.2. Variants of KLPT and sub-algorithms

7.2.1. RandomEquivalentPrimeIdeal

This concerns Algorithm 8.

Constants.

(1) `KLPT_equiv_bound_coeff`: bound on the absolute value of the coefficients of the linear combination of the small basis used to find the equivalent ideal.

(2) `KLPT_equiv_num_iter`: maximum number of trials to find an equivalent ideal of prime norm.

These constants are chosen as follows. Let $B :=$ `KLPT_equiv_num_iter`, $C :=$ `KLPT_equiv_num_iter`. We will take a value of C roughly equal to the size of the search space, so this means taking $C = (2B+1)^3(B+1) \approx 8B^4$ (we consider quaternion elements up to sign).

For a typical ideal, we have $D \approx p^{1/2}$. So this means we should choose of value of B such that

$$(1 + 2\ln 2/\log(p))^C \leq \mathsf{negl}$$

With this choice of constants, we can upper bound the bitsize of the norm of the ideals in output of the klpt_ideal_equiv algorithm by $\log_2(D) + 2\log_2(B)$.

7.2.2. FullRepresentInteger

This is about Algorithm 10. The implementation follows quite closely the algorithm FullRepresentInteger introduced as [**DLLW23**, Algorithm 1]. The only real difference is that we make γ primitive. There is a constant `KLPT_repres_num_gamma_trial` that defines the maximum number of attempts made in the algorithm FullRepresentInteger. Its value is defined by the equation below, with respect to some value `KLPT_gamma_exponent_center_shift` defined later with constants for the Signing KLPT algorithm.

$$\texttt{KLPT\_repres\_num\_gamma\_trial} = 2^{\texttt{KLPT\_gamma\_exponent\_center\_shift}}$$

7.2.3. SigningKLPT

On the ideals given in input. In the implementation, we assume that all the ideals given in input are fixed (meaning any randomization of the ideals is performed before calling the signing KLPT algorithm).

Constants.

(1) `KLPT_signing_num_gamma_trial`: number of different γ tried in signing KLPT until we try with another ideal (to rerandomize the choice of γ).
(2) `KLPT_gamma_exponent_center_shift`: shifting the exponent used to compute γ to ensure a good probability of success of the represent integer subroutine.
(3) `KLPT_signing_number_strong_approx`: number of vectors tried for the strong approximation step inside the signing KLPT algorithm.
(4) `KLPT_signing_klpt_length`: length of the output of the Signing klpt algorithm.

We fix these constants so that:

$$\begin{aligned}
&2^{-\texttt{KLPT_signing_num_gamma_trial}} \leq \mathsf{negl},\\
&-\log\left(1-\frac{1}{\log(p)}\right)\left(2^{\texttt{KLPT_gamma_exponent_center_shift}} - \texttt{KLPT_signing_num_gamma_trial}\right) \geq -\log(\mathsf{negl}),\\
&(1-4/(13\log(p)))^{\texttt{KLPT_signing_number_strong_approx}} \leq 1/64,\\
&2^{-2\cdot\texttt{KLPT_signing_klpt_length}-(15/4)\log(p)-25} \leq \mathsf{negl},
\end{aligned}$$

and `KLPT_signing_klpt_length` is a multiple of f.

7.2.4. KeyGenKLPT

Constants.

(1) `KLPT_keygen_num_gamma_trial`: number of different γ tried in keygen KLPT until we try with another ideal (to rerandomize the choice of γ).
(2) `KLPT_keygen_number_strong_approx`: number of vectors tried for the strong approximation step inside the KeyGenKLPT algorithm.
(3) `KLPT_keygen_length`: the length of the alternate secret key isogeny.

These constants can be fixed in the same way as for the KLPT signing algorithm, taking into account a different quadratic residuosity condition and the fact that the input ideal has smaller norm. In practice, we define these constants so that:

$$\begin{aligned}
&2^{\texttt{KLPT_keygen_num_gamma_trial}} \leq \mathsf{negl},\\
&(1-2/(5\log(p)))^{\texttt{KLPT_keygen_number_strong_approx}} \leq 1/64,\\
&2^{-2\cdot\texttt{KLPT_keygen_klpt_length}-(5/2)\log(p)-25} \leq \mathsf{negl},
\end{aligned}$$

and `KLPT_keygen_klpt_length` is a multiple of f.

7.2.5. SpecialEichlerNorm

On the choice of the norms N_β. Given that T has numerous primes factors, there are several ways of selecting values N_β dividing T^2. The approach taken in the implementation is the following : we select values of the form $T^2/(3^e\ell)$ where ℓ is either 1 or a non-square modulo N (depending on the quadratic residuosity condition required for N_β).

On alternate orders. A list of `KLPT_eichler_num_alternate_order` p-extremal maximal orders is precomputed for a given prime p. We select the smallest possible values of q (the only condition on q being that p must be inert in the quadratic imaginary field $\mathbb{Q}\sqrt{-q}$). Note that the case $q = 1 \mod 4$ is exactly the same (up to replacing 1 by q at a few selected places) as the case $q = 1$ which is detailed in this specification. When $q = 3 \mod 4$, the situation is slightly more complex because the order $\mathbb{Z}[\sqrt{-q}]$ is not maximal. This means that we need to divide by 2 to get a primitive element. Thus, in the implementation we include several small additional steps in the case $q = 3 \mod 4$ to ensure that the final output of SpecialEichlerNormFixed is primitive.

Constants.

(1) `KLPT_eichler_number_mu_norm`: Maximum number of target norms tried for n_μ.
(2) `KLPT_eichler_smallnorm_bitsize`: bound (in bitsize) on the norm of the ideal I given in input to the eichler norm equation for which we use the ideal I to solve the norm equation inside the Eichler order $\mathbb{Z}+I$ (and not an equivalent ideal that would a priori be bigger).
(3) `KLPT_eichler_num_equiv_ideal`: number of random equivalent ideals tried before aborting (in the generic case this bound is applied twice : once for $\mathcal{O}_0$-ideals and once for $\mathcal{O}$-ideals where $\mathcal{O}$ is an alternate special extremal order).
(4) `KLPT_eichler_number_strong_approx`: maximum number of trials for the strong approximation step inside the eichler norm algorithm.
(5) `KLPT_eichler_num_alternate_order`: number of precomputed alternate orders that we use.

These parameters are fixed as follows:

$$\texttt{KLPT\_eichler\_num\_alternate\_order} = 7$$
$$\texttt{KLPT\_eichler\_smallnorm\_bitsize} = \lceil 1/2\log(p) - 4/3(\log(T) - 5/4\log(p))$$
$$\texttt{KLPT\_eichler\_number\_mu\_norm} = \lceil (\log(T) - 5/4\log(p))/log(3)\rceil$$
$$\texttt{KLPT\_eichler\_num\_equiv\_ideal} = \lceil \log(p)/10\rceil$$
$$\texttt{KLPT\_eichler\_number\_strong\_approx} = \lceil 10\log(p)\rceil$$

7.2.6. IdealToIsogenyEichler

The IdealToIsogenyEichler$_{\ell^\bullet}$ algorithm as presented in Algorithm 19 does not exactly match the one that we implemented. In particular, for efficiency reasons, the different ideals J_i, I_i, I'_i, K_i are not always computed (when sometimes their generators is enough for the purpose). Moreover, to reduce as possible the size of the coefficients of the quaternion element we manipulate in this algorithm, we replace these ideals by equivalent ideals of smaller norm when possible. These equivalent ideals are naturally computed with the RandomEquivalentPrimeIdeal algorithm during the execution of SpecialEichlerNormFixed performed during each call to IsogenyStep.

7.2.7. Other protocol constants

(1) `KLPT_secret_key_prime_size`: bitsize of the secret key small prime. This should be $\lceil \log(p)/4\rceil$.
(2) `SQISign_signing_total_length`: A multiple of f. Equal to `KLPT_signing_klpt_length`.
(3) `SQIsign_keygen_attempts` : the maximum number of trials for the keygen computation.
(4) `SQIsign_response_attempts`: the maximum number of trials for the response phase of the signing algorithm SQIsign.Sign.

Estimating that each consecutive execution of KeyGenKLPT (resp. SigningKLPT) and $\mathsf{IdealToIsogenyEichler}_{\ell^\bullet}$ in SQIsign.KeyGen (resp. SQIsign.Sign) has at most probability $1/2$ to fail, we set `SQIsign_keygen_attempts` = `SQIsign_response_attempts` such that

$$2^{\texttt{SQIsign\_response\_attempts}} \leq \mathsf{negl}$$

CHAPTER 8

Heuristics and failure cases

SQIsign key generation and signature algorithms use the KLPT algorithm and its variants from [DKL+20, DLLW23], whose terminations depend on (plausible) heuristics. Most of the heuristics involved amount to some variation of the following assumption:

> *An integer number generated in a particular way behaves like a random number of the same size with respect to some property.*

The actual distributions involved are those occurring in the various KLPT subroutines, and will be quickly recalled below. Relevant properties are primality and quadratic residuosity.

These types of assumptions are very common in number theory, and even though some may be very hard to prove rigorously, they are usually true *"unless there is a good reason for them to be false"*, like a specific congruence property. Following [Wes22], some assumptions can be removed under the Generalized Riemann Hypothesis (GRH) but the price to pay is an increase of parameters and significant efficiency loss. SQIsign does not follow that approach.

In addition, we assume that the two-dimensional lattices appearing in the FullStrongApproximation algorithm behave like random lattices of the same discriminant, and for the SpecialEichlerNorm algorithm, we assume that β will be in K with probability $1/(\ell+1)$ if that condition was not checked.

The heuristic assumptions are used to inform decisions on parameters (see Chapter 7), to argue correctness of the overall algorithm, and to estimate runtime. More precisely, several KLPT subroutines generate integer numbers under specific distributions until they satisfy some property. In practice, the number of possible attempts is bounded to ensure termination and constant size signatures. The heuristic assumptions then allow to estimate the number of attempts a priori needed to satisfy the property, and the residual probability of failure for a given number of attempts made available.

SQIsign handles most failures in KLPT subroutines by re-randomizing these and earlier steps. This approach only reduces the overall failure probabilities; we fix an acceptable "negligible" threshold and deduce SQIsign parameters from a careful analysis.

Below, we first discuss the main assumptions and their general implications, then we review their role in the various SQIsign subroutines.

8.1. Assumptions used and their implications

8.1.1. Primality

In several SQIsign subroutines, an integer number is generated according to some specific distribution. We generally assume that the probability that these numbers are prime is identical to the probability that a random number of the same size is prime.

That is, we assume that a number of b bits is a prime with probability $\frac{-\ln(2)}{b}$.

8.1.2. Quadratic residuosity conditions

Success of strong approximation subroutine requires certain quadratic residuosity conditions to hold on numbers generated according to specific distributions. We assume that these conditions hold with the same probability as random numbers.

That is, we assume that these numbers are squares modulo N with probability 2^{-k} where k is the number of distinct prime factors of N.

8.1.3. Random lattices

Our full FullStrongApproximation subroutine searches for close vectors to a given target within a two-dimensional lattice of discriminant N. We assume that these lattices are distributed like random lattices of the same discriminant.

The distance between the closest vector in the lattice and the target vector can be approximated by λ_2, i.e. the norm of the largest vector in a reduced basis. (The actual vector selected will also need to satisfy some related primality condition, resulting in a slight increase of the distance.)

Random lattices of discriminant N are expected to satisfy $\lambda_1 \approx \lambda_2 \approx N^{1/2}$. Following [**AEN19**, Section 4.1] we can estimate the probability that the smallest vector is smaller by a factor $B > 0.6826$ compared to its expected value by

$$\int_{x=-1/2}^{1/2} \int_{y=(\frac{B}{0.6826})^2}^{\infty} \frac{3}{\pi y^2} \,\mathrm{d}x \,\mathrm{d}y = 0.4449 B^{-2}.$$

We deduce that the square norm of the CVP solution will be bigger than the expected size by a factor B with probability $0.4449 B^{-1}$.

8.1.4. Ideal sizes

Given a random ideal I, we estimate the probability that its ideal class representative with the smallest prime norm has norm larger than B by

$$\Pr[N > B] \approx \prod_{N \text{ prime}, N<B} \left(1 - \frac{N}{p/12}\right) \approx e^{-\frac{12}{p} \sum_{N \text{ prime}, N<B} N} \approx e^{-\frac{6B^2}{p \ln B}}.$$

This probability is $2^{-\lambda}$ for B such that

$$\frac{6B^2}{p \ln(B)} = \lambda.$$

In particular, adding about $\frac{1}{2} \log \lambda$ bits to the expected ideal sizes ensures that ideals norms are within the bounds with probability at least $1 - 2^{-\lambda}$. For $\lambda = 64$ this is just 3 additional bits in ideal sizes.

8.2. Overview of subroutines

8.2.1. RandomEquivalentPrimeIdeal

This algorithm randomly generates small ideal elements with random linear combinations of a reduced lattice basis, until one is found with prime reduced norm.

For analysis purposes, the norms are considered as random numbers of identical sizes. This allows to analyze success probabilities, and fix parameters such that residual failure is negligible.

Since the coefficients of the linear combination are small, the norm of the output of RandomEquivalentPrimeIdeal are related to the norm of the elements in the reduced basis of the ideal, this is related to the results detailed in Section 8.1.4.

8.2.2. SigningKLPT

This algorithm generates quaternions γ (using FullRepresentInteger) until one is found that makes subsequent steps work; in particular, a quadratic residuosity condition modulo a product of two primes must be satisfied, then a two-dimensional lattice is constructed and lattice vectors close to a target are enumerated and corresponding integers deduced, until one is found that is prime. The generation of γ elements also involves searching for primes among numbers distributed in particular way.

For analysis purposes, the quadratic residuosity test is assumed to pass with probability 1/4, and the primality conditions are assumed to be satisfied with the same probability as random numbers of the same size. The two-dimensional lattice is also modeled as a random lattice with the same discriminant. These assumptions allow to analyze success probabilities, and fix parameters such that residual failure is negligible.

8.2.3. KeyGenKLPT

Key generation uses one instance of KLPT algorithm. The analysis is similar to that of SigningKLPT, with a quadratic residuosity condition modulo a prime instead of a product of two primes, and smaller sizes.

8.2.4. SpecialEichlerNorm

The SpecialEichlerNorm algorithm applies the SpecialEichlerNormFixed on ideals generated using RandomEquivalentPrimeIdeal until one execution of SpecialEichlerNormFixed succeeds. Since FullStrongApproximation is the main sub-routine of SpecialEichlerNormFixed, the analysis of SpecialEichlerRandom combines the analysis on FullStrongApproximation and RandomEquivalentPrimeIdeal.

If N is the norm of the ideal in output of RandomEquivalentPrimeIdeal given in output to SpecialEichlerNormFixed. Then, the FullStrongApproximation will look for vectors smaller than N_β/p for some $N_\beta|T^2$ inside a lattice of dimension 2 and discriminant N^3. This means that the size of N must not be too big compared to the size of T and that the lattice must behave as a random lattice. We choose T to be big enough for this to work when $N \approx \sqrt{-p}$. If N^3 is bigger than T^2/p, then no solution can be found.

In addition, the algorithm needs to ensure that the final output is not in a certain order at distance ℓ; we heuristically assume the probability of this happening is as if the output were random, namely $(\ell+1)^{-1}$.

Using these heuristics and the estimates of Section 8.1.3 and 8.1.4, it is possible to show that for our choices of parameters, the failure probability of a given FullStrongApproximation inside SpecialEichlerNorm is big enough to ensure a negligible failure probability of SpecialEichlerNorm through randomization with alternate orders.

8.2.5. IdealToIsogenyEichler (IdealSteps)

Because the IdealToIsogenyEichler will be called during key generation and signature on isogenies whose degrees are too large to allow a direct ideal to isogeny translation, the IdealToIsogenyEichler algorithm is made of several steps, where each step involves a call to the SpecialEichlerNorm algorithm followed by a dedicated ideal to isogeny algorithm. The only source of failure is SpecialEichlerNorm.

8.2.6. Further randomization SQISIGN main algorithms

Under the assumptions listed above, there is a non-zero overall failure probability for running SigningKLPT and IdealToIsogenyEichler consecutively. This probability can be estimated as

$$\epsilon_{\mathsf{SigningKLPT}} + \left(\frac{\texttt{SQIsign\_signing\_total\_length}}{f}\right) \cdot \epsilon_{\mathsf{SpecialEichlerNorm}}$$

where $\epsilon_{\text{Signing KLPT}}$ and $\epsilon_{\text{Eichler norm}}$ are respectively the residual error probabilities in the SigningKLPT and SpecialEichlerNorm. Parameter choices for these algorithms ensure that this probability is very small.

To handle residual failures, the SQISIGN NIST submission introduces some randomization within the SigningKLPT algorithm: by applying twice the RandomEquivalentPrimeIdeal algorithm at the beginning.

A similar issue affects key generation. In that case, the randomization is ensured by the choice of the secret key.

CHAPTER 9

Security analysis

This chapter discusses the security of SQIsign. In Section 9.1, we start by proving that *weak* honest-verifier zero-knowledge and special soundness imply the security of Σ-protocols under passive impersonation attacks. Via the Fiat–Shamir heuristic, this in turn implies that the corresponding signature scheme is existentially unforgeable against chosen message attacks (EUF-CMA). We then show that the Σ-protocol used to build SQIsign satisfies the necessary security properties under certain isogeny-based hardness assumptions. In Section 9.2, we analyse the resistance of SQIsign to known attacks, which allows to tune the concrete security parameters. Finally, in Section 9.3, we conclude with a discussion on various choices made for the parameters, and how these choices have no known impact on the security.

9.1. Security reductions

9.1.1. Security of Σ-protocols

Recall the definition of a Σ-protocol from Chapter 3. We now describe properties that we want our Σ-protocol to satisfy.

9.1.1.1. Correctness. Informally, we require that if the prover knows $(x, w) \in \mathcal{R}$ and behaves honestly, then the verifier outputs 1. Formally, a Σ-protocol $(\mathsf{P}_1, \mathsf{P}_2, \mathsf{V})$ is said to be correct if for any $(x, w) \in \mathcal{R}$ and any honest stateful prover $(\mathsf{P}_1, \mathsf{P}_2)$, letting $\mathsf{com} \leftarrow \mathsf{P}_1(x, w)$, $\mathsf{chall} \leftarrow \{0,1\}^\lambda$, and $\mathsf{resp} = \mathsf{P}_2(\mathsf{chall})$, we have

$$\mathsf{V}(x, \mathsf{com}, \mathsf{chall}, \mathsf{resp}) = 1.$$

9.1.1.2. Special soundness. We recall the standard notion of special soundness for a Σ-protocol. Informally, soundness captures the guarantee that a cheating prover that *only* knows a statement x and *not* a corresponding witness w such that $(x, w) \in \mathcal{R}$ cannot force an honest verifier to accept a proof transcript. Formally, a Σ-protocol is said to satisfy n-special soundness if there exists a polynomial-time extraction algorithm that, given a statement x and n valid transcripts

$$(\mathsf{com}, \mathsf{chall}_1, \mathsf{resp}_1), \ldots, (\mathsf{com}, \mathsf{chall}_n, \mathsf{resp}_n)$$

where $\mathsf{chall}_i \neq \mathsf{chall}_j$ for all $1 \leq i < j \leq n$, outputs a witness w such that $(x, w) \in \mathcal{R}$ with probability at least $1 - \varepsilon$ for soundness error ε.

9.1.1.3. Special honest-verifier zero-knowledge (SHVZK). We also recall the standard notion of special honest-verifier zero-knowledge (SHVZK) for a Σ-protocol. We do not use this security notion in our proofs, but mention it for the sake of completeness. Informally, a Σ-protocol is said to be SHVZK if there exists a polynomial-time simulator that, given a statement x and a challenge chall, outputs a valid transcript $(\mathsf{com}, \mathsf{chall}, \mathsf{resp})$ that is indistinguishable from a real transcript.

Definition 9.1.1. A Σ-protocol $(\mathsf{P}_1, \mathsf{P}_2, \mathsf{V})$ is computationally special honest-verifier zero-knowledge if there exists a probabilistic polynomial-time simulator $\mathcal{S}$ such that for all probabilistic polynomial-time provers $(\mathsf{P}_1, \mathsf{P}_2)$ and

for all probabilistic polynomial-time stateful adversaries $\mathcal{A}$

$$\Pr\left[\mathcal{A}(\mathsf{com},\mathsf{chall},\mathsf{resp})=1 \;\middle|\; \begin{array}{l}(x,w,\mathsf{chall})\leftarrow\mathcal{A}(1^\lambda);\\ \mathsf{com}\leftarrow\mathsf{P}_1(x,w);\\ \mathsf{resp}=\mathsf{P}_2(\mathsf{chall})\end{array}\right] \approx \Pr\left[\mathcal{A}(\mathsf{com},\mathsf{chall},\mathsf{resp})=1 \;\middle|\; \begin{array}{l}(x,w,\mathsf{chall})\leftarrow\mathcal{A}(1^\lambda);\\ (\mathsf{com},\mathsf{resp})\leftarrow\mathcal{S}(x,\mathsf{chall})\end{array}\right].$$

9.1.1.4. Weak honest-verifier zero-knowledge (wHVZK). We also introduce a weaker notion of honest-verifier zero-knowledge for Σ-protocols that suffices for our security proofs. Informally, a Σ-protocol is said to satisfy *weak* honest-verifier zero-knowledge if there exists a polynomial-time simulator that, given any *uniformly randomly sampled* statements x, outputs polynomially valid transcripts of the form $(\mathsf{com},\mathsf{chall},\mathsf{resp})$ that are indistinguishable from real transcripts between an honest prover and an honest verifier. We define this notion formally below.

DEFINITION 9.1.2. A Σ-protocol $(\mathsf{P}_1,\mathsf{P}_2,\mathsf{V})$ is computationally weak honest-verifier zero-knowledge (wHVZK) with respect to a uniform instance generation algorithm Gen if for any $Q=\mathrm{poly}(\lambda)$, for all probabilistic polynomial-time stateful provers $(\mathsf{P}_1,\mathsf{P}_2)$ and for all probabilistic polynomial-time (honest) verifiers V, there exists a probabilistic polynomial-time simulator $\mathcal{S}$ such that for all probabilistic polynomial-time distinguishers $\mathcal{D}$

$$\Pr\left[\mathcal{D}\left(x,\{\mathsf{com}_i,\mathsf{chall}_i,\mathsf{resp}_i\}_{i\in[1,Q]}\right)=1 \;\middle|\; \begin{array}{l}(x,w)\leftarrow\mathsf{Gen}(1^\lambda);\\ \{\mathsf{com}_i\leftarrow\mathsf{P}_1(x,w);\\ \mathsf{chall}_i\leftarrow\{0,1\}^\lambda;\\ \mathsf{resp}_i=\mathsf{P}_2(\mathsf{chall}_i)\}_{i\in[1,Q]}\end{array}\right] \approx \Pr\left[\mathcal{D}\left(x,\{\mathsf{com}_i,\mathsf{chall}_i,\mathsf{resp}_i\}_{i\in[1,Q]}\right)=1 \;\middle|\; \begin{array}{l}(x,w)\leftarrow\mathsf{Gen}(1^\lambda);\\ \{(\mathsf{com}_i,\mathsf{chall}_i,\mathsf{resp}_i)\leftarrow\mathcal{S}(x)\}_{i\in[1,Q]}\end{array}\right].$$

REMARK 1. We note that the notion of wHVZK security as defined above is strictly weaker than the notion of SHVZK security defined earlier as: (a) the wHVZK notion only holds for uniformly sampled instances (as opposed to adversarially chosen instances in the case of SHVZK), and (b) the distinguisher in the wHVZK security game is not provided with the witness (as opposed to the SHVZK security game where the distinguishing adversary knows the witness).

9.1.1.5. Security under passive impersonation attacks (IMP-PA). Finally, we recall the notion of security under passive impersonation attacks (IMP-PA) for any Σ-protocol (introduced originally in [**AABN02**]). Informally, security against impersonation under passive attacks considers an adversary – here called an impersonator – whose goal is to impersonate an honest prover on a uniformly random statement without the knowledge of the corresponding secret witness. As outlined in [**AABN02**], such an adversary generally has access not only to the statement but also to conversations between the real prover and an honest verifier, possibly via eavesdropping over the network.

Following the definition of IMP-PA security in [**AABN02**], we model this setting by viewing an impersonator as a probabilistic algorithm $\mathcal{I}$ and giving to it the statement and the transcript-generation oracle defined above. This oracle gives the impersonator $\mathcal{I}$ the ability to obtain some number of transcripts of honest executions of the protocol. After reviewing the transcripts, the impersonator must then participate in the Σ-protocol with an honest verifier and try to get the verifier to accept. We now recall the formal definition of IMP-PA security from [**AABN02**] below.

DEFINITION 9.1.3. A Σ-protocol $(\mathsf{P}_1,\mathsf{P}_2,\mathsf{V})$ satisfies security under passive impersonation attacks (IMP-PA) with respect to a uniform instance generation algorithm Gen if for all probabilistic polynomial-time cheating (passive) impersonators $\mathcal{I}=(\mathcal{I}_1,\mathcal{I}_2)$ (where the impersonator has access to a randomized transcript generation oracle TR defined subsequently), we have

$$\Pr\left[\mathsf{V}\left(x,\mathsf{com},\mathsf{chall},\mathsf{resp}\right)=1 \;\middle|\; \begin{array}{l}(x,w)\leftarrow\mathsf{Gen}(1^\lambda);\\ (\mathsf{st},\mathsf{com})\leftarrow\mathcal{I}_1^{\mathsf{TR}_{(x,w)}()}(x);\\ \mathsf{chall}\leftarrow\{0,1\}^\lambda;\\ \mathsf{resp}\leftarrow\mathcal{I}_2(\mathsf{st},\mathsf{chall})\end{array}\right]\leq\mathsf{negl}(\lambda),$$

where $\mathsf{TR}_{(x,w)}()$ is a randomized transcript generation oracle which takes no inputs and returns a random transcript of an "honest" execution of the Σ-protocol w.r.t. (x, w), namely:

$$\begin{array}{l}\underline{\mathsf{TR}_{(x,w)}() :}\\ \mathsf{com} \leftarrow \mathsf{P}_1(x, w);\\ \mathsf{chall} \leftarrow \{0,1\}^\lambda;\\ \mathsf{resp} = \mathsf{P}_2(\mathsf{chall})\end{array}$$

9.1.1.6. Special-soundness + wHVZK $\implies$ IMP-PA. We now state and prove the following theorem.

THEOREM 9.1.4. *Any Σ-protocol $(\mathsf{P}_1, \mathsf{P}_2, \mathsf{V})$ that satisfies special-soundness and wHVZK also satisfies IMP-PA.*

To prove this theorem, we first introduce a modified version of the transcript-generation oracle defined above that generates random "simulated" transcripts of an "honest" execution given only the statement x, namely:

$$\begin{array}{l}\underline{\widehat{\mathsf{TR}}_{(x)}() :}\\ (\mathsf{com}, \mathsf{chall}, \mathsf{resp}) \leftarrow \mathcal{S}_{\text{wHVZK}}(x)\end{array}$$

where $\mathcal{S}_{\text{wHVZK}}$ is the simulator for the Σ-protocol as per the wHVZK security definition. We now state and prove the following propositions.

PROPOSITION 9.1.5. *Let $(\mathsf{P}_1, \mathsf{P}_2, \mathsf{V})$ be a sigma protocol that satisfies wHVZK. Suppose that there exists some probabilistic polynomial-time cheating (passive) impersonator $\mathcal{I} = (\mathcal{I}_1, \mathcal{I}_2)$ such that*

$$\Pr\left[\mathsf{V}(x, \mathsf{com}, \mathsf{chall}, \mathsf{resp}) = 1 \;\middle|\; \begin{array}{l}(x, w) \leftarrow \mathsf{Gen}(1^\lambda);\\ (\mathsf{st}, \mathsf{com}) \leftarrow \mathcal{I}_1^{\mathsf{TR}_{(x,w)}()}(x);\\ \mathsf{chall} \leftarrow \{0,1\}^\lambda;\\ \mathsf{resp} \leftarrow \mathcal{I}_2(\mathsf{st}, \mathsf{chall})\end{array}\right] = \varepsilon.$$

Then, for all probabilistic polynomial-time cheating (passive) impersonators $\widehat{\mathcal{I}} = (\widehat{\mathcal{I}}_1, \widehat{\mathcal{I}}_2)$ with access to the "simulated" transcript generation oracle $\widehat{\mathsf{TR}}$ as defined above, we must have

$$\Pr\left[\mathsf{V}(x, \mathsf{com}, \mathsf{chall}, \mathsf{resp}) = 1 \;\middle|\; \begin{array}{l}(x, w) \leftarrow \mathsf{Gen}(1^\lambda);\\ (\mathsf{st}, \mathsf{com}) \leftarrow \widehat{\mathcal{I}}_1^{\widehat{\mathsf{TR}}_x()}(x);\\ \mathsf{chall} \leftarrow \{0,1\}^\lambda;\\ \mathsf{resp} \leftarrow \widehat{\mathcal{I}}_2(\mathsf{st}, \mathsf{chall})\end{array}\right] \leq \varepsilon + \mathsf{negl}(\lambda).$$

PROOF. We prove this by contradiction. Suppose that there exists a cheating (passive) impersonator $\widehat{\mathcal{I}} = (\widehat{\mathcal{I}}_1, \widehat{\mathcal{I}}_2)$ such that

$$\Pr\left[\mathsf{V}(x, \mathsf{com}, \mathsf{chall}, \mathsf{resp}) = 1 \;\middle|\; \begin{array}{l}(x, w) \leftarrow \mathsf{Gen}(1^\lambda);\\ (\mathsf{st}, \mathsf{com}) \leftarrow \widehat{\mathcal{I}}_1^{\widehat{\mathsf{TR}}_x()}(x);\\ \mathsf{chall} \leftarrow \{0,1\}^\lambda;\\ \mathsf{resp} \leftarrow \widehat{\mathcal{I}}_2(\mathsf{st}, \mathsf{chall})\end{array}\right] > \varepsilon + \mathsf{negl}(\lambda).$$

Then it is easy to see that we can use the impersonator $\mathcal{I}$ in the original IMP-PA security game (the "real" transcript generation oracle $\mathsf{TR}_{(x,w)}()$) and the impersonator $\widehat{\mathcal{I}}$ in the modified IMP-PA security game (with the "simulated" transcript generation oracle $\widehat{\mathsf{TR}}_x()$) to construct a probabilistic polynomial distinguisher $\mathcal{D}$ that distinguishes (with non-negligible probability) the outputs of the "real" transcript generation oracle $\mathsf{TR}_{(x,w)}()$ and the "simulated" transcript generation oracle $\widehat{\mathsf{TR}}_x()$. This immediately contradicts the assumption that Σ satisfies wHVZK. This completes the proof of Proposition 9.1.5. □

PROPOSITION 9.1.6. *Let* $(\mathsf{P}_1, \mathsf{P}_2, \mathsf{V})$ *be a sigma protocol that satisfies special-soundness. Then, for all probabilistic polynomial-time cheating (passive) impersonators* $\widehat{\mathcal{I}} = (\widehat{\mathcal{I}}_1, \widehat{\mathcal{I}}_2)$ *with access to the "simulated" transcript generation oracle* $\widehat{\mathsf{TR}}$ *defined above, we must have*

$$\Pr\left[\mathsf{V}\left(x, \mathsf{com}, \mathsf{chall}, \mathsf{resp}\right) = 1 \;\middle|\; \begin{array}{l} (x, w) \leftarrow \mathsf{Gen}(1^\lambda); \\ (\mathsf{st}, \mathsf{com}) \leftarrow \widehat{\mathcal{I}}_1^{\widehat{\mathsf{TR}}_x()}(x); \\ \mathsf{chall} \leftarrow \{0,1\}^\lambda; \\ \mathsf{resp} \leftarrow \widehat{\mathcal{I}}_2(\mathsf{st}, \mathsf{chall}) \end{array}\right] \leq \mathsf{negl}(\lambda).$$

PROOF. We again prove this by contradiction. Suppose that there exists a cheating (passive) impersonator $(\widehat{\mathcal{I}}_1, \widehat{\mathcal{I}}_2)$

$$\Pr\left[\mathsf{V}\left(x, \mathsf{com}, \mathsf{chall}, \mathsf{resp}\right) = 1 \;\middle|\; \begin{array}{l} (x, w) \leftarrow \mathsf{Gen}(1^\lambda); \\ (\mathsf{st}, \mathsf{com}) \leftarrow \widehat{\mathcal{I}}_1^{\widehat{\mathsf{TR}}_x()}(x); \\ \mathsf{chall} \leftarrow \{0,1\}^\lambda; \\ \mathsf{resp} \leftarrow \widehat{\mathcal{I}}_2(\mathsf{st}, \mathsf{chall}) \end{array}\right] > \mathsf{negl}(\lambda).$$

Then we can use this impersonator $\widehat{\mathcal{I}}$ to construct a probabilistic polynomial adversary $\mathcal{A}$ that forges (with non-negligible probability) a valid transcript corresponding to an honest execution of Σ between a prover and a verifier on an instance (x, w) given only the statement x. Note that the existence of such an adversary $\mathcal{A}$ immediately contradicts the assumption that Σ satisfies special-soundness.

We construct the adversary $\mathcal{A}$ as follows. The adversary $\mathcal{A}$, on input x, acts as the challenger in the modified IMP-PA security game and proceeds as described below:

- On receipt of an oracle query from the impersonator $(\widehat{\mathcal{I}}_1, \widehat{\mathcal{I}}_2)$, the adversary $\mathcal{A}$ outputs a simulated transcript
$$(\mathsf{com}, \mathsf{chall}, \mathsf{resp}) \leftarrow \mathcal{S}_{\text{wHVZK}}(x).$$
- Eventually the impersonator algorithm $\widehat{\mathcal{I}}_1$ outputs a commitment com^*. The adversary $\mathcal{A}$ poses as a honest verifier and outputs a challenge $\mathsf{chall}^* \leftarrow \{0,1\}^\lambda$.
- Finally, the impersonator algorithm $\widehat{\mathcal{I}}_2$ outputs a response resp^*. At this point the adversary $\mathcal{A}$ outputs the following forged transcript:
$$\left(\mathsf{com}^*, \mathsf{chall}^*, \mathsf{resp}^*\right).$$

Note that the view of the cheating impersonator $(\widehat{\mathcal{I}}_1, \widehat{\mathcal{I}}_2)$ in the above game is identical to that in the modified IMP-PA security game. This completes the proof of Proposition 9.1.6. □

Note that propositions 9.1.5 and 9.1.6 immediately yield the following corollary.

COROLLARY 9.1.7. *Let* $(\mathsf{P}_1, \mathsf{P}_2, \mathsf{V})$ *be a sigma protocol that satisfies special-soundness and wHVZK. Then, for all cheating (passive) impersonators* $\mathcal{I} = (\mathcal{I}_1, \mathcal{I}_2)$*, we must have*

$$\Pr\left[\mathsf{V}\left(x, \mathsf{com}, \mathsf{chall}, \mathsf{resp}\right) = 1 \;\middle|\; \begin{array}{l} (x, w) \leftarrow \mathsf{Gen}(1^\lambda); \\ (\mathsf{st}, \mathsf{com}) \leftarrow \mathcal{I}_1^{\mathsf{TR}_{(x,w)}()}(x); \\ \mathsf{chall} \leftarrow \{0,1\}^\lambda; \\ \mathsf{resp} \leftarrow \mathcal{I}_2(\mathsf{st}, \mathsf{chall}) \end{array}\right] < \mathsf{negl}(\lambda).$$

The proof of Theorem 9.1.4 is immediate from Corollary 9.1.7 (in fact, Corollary 9.1.7 is merely a re-statement of Theorem 9.1.4). □

9.1.2. Digital signatures and the Fiat-Shamir Transform

In this section, we recall the standard notion of security for digital signatures (namely, existential unforgeability against chosen message attacks, or EUF-CMA in short) and the standard transformation from sigma protocols to digital signatures via the Fiat-Shamir transform/heuristic [**FS87**, **PS00**].

9.1.2.1. EUF-CMA security of digital signatures. We begin by recalling the notion of existential unforgeability against chosen message attacks (EUF-CMA) for a digital signature scheme. We define the security notion in the presence of a random oracle $H : \{0,1\}^* \to \{0,1\}^\lambda$, since this is the notion of security that we will eventually prove for our SQISIGN signature scheme (in fact, this is the standard notion of EUF-CMA security for any digital signature scheme obtained via applying the Fiat-Shamir transform on a Σ-protocol).

DEFINITION 9.1.8. A digital signature scheme $\Gamma = (\mathsf{Gen}_{\mathsf{Sig}}, \mathsf{Sign}, \mathsf{Ver})$ is said to satisfy EUF-CMA security if for all probabilistic polynomial-time forging adversaries $\mathcal{A}$ (where $\mathcal{A}$ has access to a chosen-message signature generation oracle and a random oracle H as described above), we have

$$\Pr\left[\mathsf{Ver}\left(\mathsf{pk}, \mathsf{msg}^*, \sigma^*\right) = 1 \;\middle|\; \begin{array}{l} (\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{Gen}_{\mathsf{Sig}}(1^\lambda); \\ (\mathsf{msg}^*, \sigma^*) \leftarrow \mathcal{A}^{\mathsf{Sign}(\mathsf{sk},\cdot),H(\cdot)}(\mathsf{pk}) \end{array}\right] \leq \mathsf{negl}(\lambda),$$

subject to the restriction that the adversary $\mathcal{A}$ has *not* issued a signing oracle query on the message msg^*.

9.1.2.2. From IMP-PA security to EUF-CMA security. We additionally recall the following theorem from [AABN02].

THEOREM 9.1.9 (Imported from [AABN02]). *Let $\Pi_\Sigma = (\mathsf{P}_1, \mathsf{P}_2, \mathsf{V})$ be a Σ-protocol, and let $\Gamma = (\mathsf{Gen}_{\mathsf{Sig}}, \mathsf{Sign}, \mathsf{Ver})$ be a digital signature scheme obtained from Π_Σ via the Fiat-Shamir transform as described above. If Π_Σ satisfies IMP-PA security, then Γ satisfies EUF-CMA security (in the random oracle model).*

We refer to [AABN02] for the detailed proof of Theorem 9.1.9. We now state the following corollary of Theorems 9.1.4 and 9.1.9.

COROLLARY 9.1.10. *Let $\Pi_\Sigma = (\mathsf{P}_1, \mathsf{P}_2, \mathsf{V})$ be a Σ-protocol, and let $\Gamma = (\mathsf{Gen}_{\mathsf{Sig}}, \mathsf{Sign}, \mathsf{Ver})$ be a digital signature scheme obtained from Π_Σ via the Fiat-Shamir transform as described above. If Π_Σ satisfies special-soundness and wHVZK, then Γ satisfies EUF-CMA security (in the random oracle model).*

9.1.3. Security of SQISIGN

In this section, we use the aforementioned theorems related to IMP-PA security of Σ-protocols, EUF-CMA security of digital signature schemes, and the Fiat-Shamir transform to establish the EUF-CMA security of the SQISIGN signature scheme (in the random oracle model).

9.1.3.1. Special soundness of the SQISIGN Σ-protocol. We summarize the special soundness guarantees of the SQISIGN Σ-protocol below. We begin by recalling the supersingular smooth endomorphism problem, which is the hard problem underlying the special soundness guarantee of the SQISIGN Σ-protocol.

PROBLEM 9.1.11 (Supersingular Smooth Endomorphism Problem). Given a prime p and a supersingular elliptic curve E over $\mathbb{F}_{p^2}$, find a (non-trivial) cyclic endomorphism of E of smooth degree.

The following theorem (imported from [DKL+20]) formally captures the special soundness guarantee of the SQISIGN Σ-protocol.

THEOREM 9.1.12 (Special Soundness of SQISIGN Σ-protocol [DKL+20]). *If there is a probabilistic polynomial adversary that breaks the soundness of the protocol with probability w and expected running time r for the public key E_A, then there is an algorithm for the Supersingular Smooth Endomorphism Problem on E_A with expected running time $O(r/(w - 1/c))$, where c is the size of the challenge space.*

We refer the readers to [DKL+20] for the detailed proof and associated discussions.

9.1.3.2. wHVZK of the SQISIGN Σ-protocol. Informally, wHVZK of the SQISIGN Σ-protocol relies on the assumption that the distribution of responses is computationally indistinguishable from random isogenies of the same degree starting from the curve E_A. Following [DKL+20, Proposition 11], this assumption follows from three other assumptions which can be studied separately.

- The first assumption (Assumption 1 in [DKL+20]) essentially states that the SQISIGN signing algorithm succeeds with overwhelming probability for the parameter choices taken.

- In [DKL+20] it is shown that every SQIsign signature is the pushover of some isogeny ι in a certain set $\mathcal{P}_{N_{D_{\text{secret}}}}$ (Definition 2 in [DKL+20]), and moreover this set only depends on $N_{D_{\text{secret}}}$ (i.e. not on the particular secret isogeny). The second assumption (Assumption 2 in [DKL+20]) is equivalent to assuming that for uniformly random inputs to SigningKLPT algorithm, the distribution of the isogeny ι is statistically close to uniform in $\mathcal{P}_{N_{D_{\text{secret}}}}$.
- Finally, the third assumption (hardness of Problem 2 in [DKL+20]) states that the pushover of ι through τ is computationally indistinguishable from a random isogeny of the same degree starting from E_A, even after a polynomial number of queries to an oracle sampling elements from the first distribution.

The proof of Proposition 11 in [DKL+20] shows that these three assumptions together imply weak honest verifier zero-knowledge. We note that [DKL+20] incorrectly claims that the Σ-protocol underlying SQIsign is honest verifier zero-knowledge; this is not true because the distinguishing problem is easy given the secret key, but the incorrect proof they provide actually proves *weak* honest verifier zero-knowledge. We also note that the second assumption above, although statistical in [DKL+20], could be relaxed to a computational assumption with trivial changes to the proof.

We refer to [DKL+20] for additional discussion and formal statements of the first two assumptions, and only reproduce Problem 2 from [DKL+20] below. Here, $\texttt{Iso}_{D,j(E)}$ represents the set of cyclic isogenies of degree D, whose domain is a curve inside the isomorphism class of E.

Problem 9.1.13. Let p be a prime and D a smooth integer. Let $\tau : E_0 \to E_A$ be a random isogeny drawn from $\mathcal{K}$, and let N_τ be its degree. Let $\mathcal{P}_{N_\tau} \subset \texttt{Iso}_{D,j_0}$ as in [DKL+20, Definition 2], and let O_τ be an oracle sampling random elements in $[\tau]_*\mathcal{P}_{N_\tau}$. Let $\sigma : E_A \to \star$ of degree D where either

(1) σ is uniformly random in $\texttt{Iso}_{D,j(E_A)}$;
(2) σ is uniformly random in $[\tau]_* \mathcal{P}_{N_\tau}$.

The problem is, given $p, D, \mathcal{K}, E_A, \sigma$, to distinguish between the two cases with a polynomial number of queries to O_τ.

9.1.3.3. EUF-CMA security of SQIsign. Since the SQIsign Σ-protocol satisfies both special soundness and wHVZK, it follows immediately from Corollary 9.1.10 that the SQIsign signature scheme satisfies EUF-CMA security. We capture this in the following theorem.

Theorem 9.1.14. *Assuming the computational hardness of problems 9.1.11 and 9.1.13, the SQIsign signature scheme satisfies EUF-CMA security under the Fiat-Shamir heuristic.*

9.2. Resistance to known attacks

SQIsign parameters are selected to satisfy the following requirements (amongst others – see Chapter 9):

- The primes p are chosen such that $\log p \approx 2\lambda$.
- The secret isogeny is chosen with degree a prime bounded by B, where B is chosen so that the number of such isogenies is close to $p^{1/2}$.
- The commitment isogeny degree D_{com} is approximately $2^{2\lambda} \approx p$.
- The challenge isogeny degree D_{chall} is approximately $2^\lambda \approx p^{1/2}$.

Below, we discuss known attacks on SQIsign and show that these parameter choices offer at least λ bits of security against all classical attacks and $\lambda/2$ bits of security against all quantum attacks. Throughout, for any function f, we will write $\widetilde{O}(f)$ to mean $O(f\ell(f))$, where ℓ is any polylogarithmic function. Note that for several attacks below, these logarithmic factors will contribute to more than a few bits of additional security.

9.2.1. Endomorphism ring and general isogeny computation

An algorithm to compute the endomorphism ring of a supersingular elliptic curve would break essentially all isogeny-based cryptography schemes, including SQIsign [EHL+18]. Indeed, from the endomorphism ring one can compute a corresponding maximal order in the quaternion algebra, and from there compute an isogeny between E_0 and E_A, which can then act as an equivalent secret key.

The endomorphism ring of a supersingular curve can be computed in time $O\left((\log p)^2 p^{1/2}\right) \subseteq \widetilde{O}(p^{1/2})$ and low memory requirements using the algorithm from [**EHL+20**]. Using a quantum computer and Grover search, the time becomes $\widetilde{O}(p^{1/4})$, still with low memory requirements.

For supersingular curves, the general isogeny problem (i.e., given two curves, finding an isogeny of arbitrary degree between them) is equivalent to the endomorphism computation problem [**EHL+18, Wes22**], hence $\widetilde{O}(p^{1/2})$ classically and $\widetilde{O}(p^{1/4})$ quantumly.

9.2.2. Key recovery

In SQIsign, the secret isogeny has secret degree a prime bounded by B, where B is chosen so that the number of such isogenies is about $p^{1/2}$. Note that the equivalence result between endomorphism ring computation and isogeny computation does not apply here because of the degree bound. One could try exhaustive search instead; with our parameter choices this is just as costly as endomorphism ring computation.

One can also recover a secret key in SQIsign by first computing the challenge isogeny between E_0 and E_1. Indeed with that isogeny, the challenge isogeny and the signature isogeny, one deduces an "equivalent key", namely an isogeny from E_0 to E_A, which is sufficient to sign. As D_{com} is smooth, a meet-in-the-middle strategy will also cost about $\widetilde{O}(p^{1/2})$ classically, but with higher memory costs compared to other attacks. Quantumly, the cost is less clear but well above $\widetilde{O}(p^{1/4})$. The best approach to compute the challenge isogeny seems to involve first computing the endomorphism ring of E_1 with the memory free algorithms mentioned above.

9.2.3. Soundness / forgery attacks

There is a trivial attack consisting in generating a random isogeny of the expected degree $\sigma : E_A \to E_2$ for the signature, a random isogeny $\hat{\varphi} : E_2 \to E_1$ of the expected degree for the challenge, and hoping that the hash of E_1 is consistent with the challenge isogeny. Since $D_{\text{chall}} \approx p^{1/2}$, there are about as many challenge degree isogenies. The cost of this attack is $\widetilde{O}(p^{1/2})$ classically and $\widetilde{O}(p^{1/4})$ quantumly.

By [**DKL+20**, Theorem 1], breaking soundness reduces to computing one non trivial endomorphism of E_A. Here, heuristically, the distribution of E_A is statistically close to uniform (since the secret isogeny φ_{secret} is essentially uniform among isogenies of $\lambda/2$-bit prime degree). The best known attack for this task seems to be the subroutine used in the endomorphism ring computation algorithm of [**EHL+18**], with roughly the same classical and quantum complexities, namely $\widetilde{O}(p^{1/2})$ classically and $\widetilde{O}(p^{1/4})$ quantumly.

9.2.4. Zero-knowledge

Zero-knowledge of the underlying identification protocol relies on the assumption that the distribution of signatures is computationally indistinguishable from random isogenies of the same degree from the curve E_A. Note that this assumption is computational: with the secret isogeny one can trivially distinguish signatures from random isogenies of the same degree originating at E_A. As argued above, the cost of computing the secret isogeny is $\widetilde{O}(p^{1/2})$ classically and $\widetilde{O}(p^{1/4})$ quantumly.

This is an ad hoc assumption, introduced in [**DKL+20**]. Tentative algorithms to solve the distinguishing problem are also discussed in [**DKL+20, DLLW23**]. We summarize some observations from there:

- A signature isogeny is the pushover through the secret isogeny of an isogeny in some set $\mathcal{P}_{N_{D_{\text{secret}}}}$ (Definition 2 in [**DKL+20**]), where this set $\mathcal{P}_{N_{D_{\text{secret}}}}$ only depends on D_{secret}.
- Polynomially small sets $\mathcal{P}_{N_{D_{\text{secret}}}}$ easily lead to distinguishing attacks, as one can then query signatures until observing a collision (when D_{secret} is known, this would even lead to an efficient key recovery attack). More generally, one can note that $|\mathcal{P}_{N_{D_{\text{secret}}}}|^{1/2}$ signature oracle calls will lead to two identical signatures by the birthday paradox, whereas such a collision is only expected to appear after $D_{sig}^{1/2} \gg p^{5/8}$ for random isogenies from E_A.
- The size of $\mathcal{P}_{N_{D_{\text{secret}}}}$ is $\widetilde{O}(D_{\text{secret}} \cdot p)$ (see [**DKL+20**, Remark 22]) hence the cost of the distinguishing attack would be at least $\widetilde{O}(p^{5/8})$.

- The original SQIsign algorithm from [DKL+20] computed γ in $\mathbb{Z}+\mathbb{Z}\,i+\mathbb{Z}\,j+\mathbb{Z}\,k$ instead of O_0, therefore introducing a bias in the signature distribution. This bias led to a distinguishing attack against the assumption, which is countered by "properly" generating γ [DLLW23].

It can be noted that an attack against this assumption does not immediately imply an attack against the signature scheme, as shown with the distinguishing attack on the original SQIsign scheme from [DLLW23].

9.2.5. Algorithm failures and their impact on security

There are numerous failure cases to the various subroutines used in SQIsign. Some happen with constant probability (modularity conditions, last step in Eichler norm equation algorithm), some with small but non-negligible probability (success in the strong approximation subroutine), and some with negligible probability (for example, a random ideal having an exceptionally large minimal prime representative). The failure cases occurring with non-negligible probability are mitigated by re-randomizing previous steps, and our parameter selection ensures that the overall failure probability is negligible (under assumptions listed in Chapter 8). As a consequence, failures are not expected to affect the unforgeability of SQIsign.

In a timing attack model or more generally given side-channel information, an adversary may be able to identify internal failures to the algorithm and deduce secret information. We consider these attack models beyond the scope of this submission, and we therefore do not discuss them further here. However, it is a matter of interest for future versions, as the community is actively investigating this direction [JMRK23].

9.3. Nothing upon my sleeve

Here we provide some additional comments on our choices of parameters, arguing that these either have no impact on security, or at least no known impact on security.

- We restrict our parameters to primes p congruent to 3 modulo 4. One could define SQIsign without that restriction but as argued in Chapter 5, this choice significantly simplifies certain computations. Note that is a standard choice in the literature, ensuring that 1728 is the j-invariant of a supersingular elliptic curve. This choice only reduces the set of potential parameters by one half, and we are not aware that it affects any of the computational assumptions SQIsign relies upon.
- We choose the curve E_0 with j-invariant $j = 1728$. One could define SQIsign using any arbitrary curve as a starting curve E_0, but this would significantly affect efficiency. Indeed the signing algorithm gets improved from the existence of a "small" degree isogeny between E_A and a curve with a small non trivial endomorphism. From a security point of view, replacing the curve E_0 by any arbitrary supersingular curve (with endomorphism ring known to the signer, as this is crucial to compute signatures) seems to have no benefit. We are not aware that knowledge of one endomorphism ring would give any significant advantage in computing the secret isogeny between E_0 and E_A (speedups could be obtained using automorphisms in E_0 in the context of SIKE, but those speedups are tiny). The other best attacks seem to involve computing at least one endomorphism of a statistically random curve, and are indifferent to the choice of starting curve.
- We put very strong smoothness constraints on p^2-1 for the primes used. As we argue in Chapter 5, this choice is motivated by efficiency considerations, namely to avoid computations over field extensions of $\mathbb{F}_{p^2}$. This approach to optimizing the field arithmetic has long been standard in classical elliptic curve cryptography, and more recently in isogeny-based cryptography (CSIDH, SIDH and their variants all use primes of some special form), and has never been connected to any weakness. More generally, the primes we choose satisfy multiple constraints for efficiency reasons, as detailed in Chapter 5. We are not aware of any cryptanalysis approach to exploit the particular shape of the prime, beyond polynomial speedups provided by the faster arithmetic over $\mathbb{F}_{p^2}$.

Bibliography

[AABN02] Michel Abdalla, Jee Hea An, Mihir Bellare, and Chanathip Namprempre. From identification to signatures via the Fiat-Shamir transform: Minimizing assumptions for security and forward-security. In Lars R. Knudsen, editor, *EUROCRYPT 2002*, volume 2332 of *LNCS*, pages 418–433. Springer, Heidelberg, April / May 2002. doi:10.1007/3-540-46035-7_28.

[ACR23] Gora Adj, Jesús-Javier Chi-Domínguez, and Francisco Rodríguez-Henríquez. Karatsuba-based square-root Vélu's formulas applied to two isogeny-based protocols. *J. Cryptogr. Eng.*, 13(1):89–106, 2023. doi:10.1007/s13389-022-00293-y.

[AEN19] Yoshinori Aono, Thomas Espitau, and Phong Q. Nguyen. Random lattices: Theory and practice, 2019. URL: https://espitau.github.io/bin/random_lattice.pdf.

[ARH14] Gora Adj and Francisco Rodríguez-Henríquez. Square root computation over even extension fields. *IEEE Transactions on Computers*, 63(11):2829–2841, 2014. doi:10.1109/TC.2013.145.

[Bab86] László Babai. On Lovász' lattice reduction and the nearest lattice point problem. *Combinatorica*, 6(1):1–13, 1986.

[BDLS20] Daniel J Bernstein, Luca De Feo, Antonin Leroux, and Benjamin Smith. Faster computation of isogenies of large prime degree. *Open Book Series*, 4(1):39–55, 2020.

[Ber04] Daniel J. Bernstein. How to find smooth parts of integers, 2004. http://cr.yp.to/papers.html#smoothparts.

[BSC+22] Giacomo Bruno, Maria Corte-Real Santos, Craig Costello, Jonathan Komada Eriksen, Michael Naehrig, Michael Meyer, and Bruno Sterner. Cryptographic smooth neighbors. Cryptology ePrint Archive, Report 2022/1439, 2022. https://eprint.iacr.org/2022/1439.

[CD23] Wouter Castryck and Thomas Decru. An efficient key recovery attack on SIDH. In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part V*, volume 14008 of *LNCS*, pages 423–447. Springer, Heidelberg, April 2023. doi:10.1007/978-3-031-30589-4_15.

[CH17] Craig Costello and Hüseyin Hisil. A simple and compact algorithm for SIDH with arbitrary degree isogenies. In Tsuyoshi Takagi and Thomas Peyrin, editors, *ASIACRYPT 2017, Part II*, volume 10625 of *LNCS*, pages 303–329. Springer, Heidelberg, December 2017. doi:10.1007/978-3-319-70697-9_11.

[CLN16] Craig Costello, Patrick Longa, and Michael Naehrig. Efficient algorithms for supersingular isogeny Diffie-Hellman. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part I*, volume 9814 of *LNCS*, pages 572–601. Springer, Heidelberg, August 2016. doi:10.1007/978-3-662-53018-4_21.

[CMN21] Craig Costello, Michael Meyer, and Michael Naehrig. Sieving for twin smooth integers with solutions to the prouhet-tarry-escott problem. In Anne Canteaut and François-Xavier Standaert, editors, *EUROCRYPT 2021, Part I*, volume 12696 of *LNCS*, pages 272–301. Springer, Heidelberg, October 2021. doi:10.1007/978-3-030-77870-5_10.

[Coh93] Henri Cohen. *A Course in Computational Algebraic Number Theory*. Springer, New York, NY, USA, 1993. doi:10.1007/978-3-662-02945-9.

[Cor08] Giuseppe Cornacchia. Su di un metodo per la risoluzione in numeri interi dell'equazione $\sum_{h=0}^{n} c_h x^{n-h} y^h = p$. *Giornale di Matematiche di Battaglini*, 46:33–90, 1908.

[Cos20] Craig Costello. B-SIDH: Supersingular isogeny Diffie-Hellman using twisted torsion. In Shiho Moriai and Huaxiong Wang, editors, *ASIACRYPT 2020, Part II*, volume 12492 of *LNCS*, pages 440–463. Springer, Heidelberg, December 2020. doi:10.1007/978-3-030-64834-3_15.

[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, September 2018. doi:10.1007/s13389-017-0157-6.

[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, *ASIACRYPT 2020, Part I*, volume 12491 of *LNCS*, pages 64–93. Springer, Heidelberg, December 2020. doi:10.1007/978-3-030-64837-4_3.

[DLLW23] Luca De Feo, Antonin Leroux, Patrick Longa, and Benjamin Wesolowski. New algorithms for the deuring correspondence - towards practical and secure SQISign signatures. In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part V*, volume 14008 of *LNCS*, pages 659–690. Springer, Heidelberg, April 2023. doi:10.1007/978-3-031-30589-4_23.

[Ebe08] David Eberly. The Laplace expansion theorem: Computing the determinants and inverses of matrices, 2008. URL: https://www.geometrictools.com/Documentation/LaplaceExpansionTheorem.pdf.

[EHL+18] Kirsten Eisenträger, Sean Hallgren, Kristin E. Lauter, Travis Morrison, and Christophe Petit. Supersingular isogeny graphs and endomorphism rings: Reductions and solutions. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part III*, volume 10822 of *LNCS*, pages 329–368. Springer, Heidelberg, April / May 2018. doi:10.1007/978-3-319-78372-7_11.

[EHL+20] Kirsten Eisenträger, Sean Hallgren, Chris Leonardi, Travis Morrison, and Jennifer Park. Computing endomorphism rings of supersingular elliptic curves and connections to path-finding in isogeny graphs. *Open Book Series*, 4(1):215–232, 2020.

[FP85] Ulrich Fincke and Michael Pohst. Improved methods for calculating vectors of short length in a lattice, including a complexity analysis. *Mathematics of Computation*, 44(170):463–471, 1985. doi:10.1090/S0025-5718-1985-0777278-8.

[FS87] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *CRYPTO'86*, volume 263 of *LNCS*, pages 186–194. Springer, Heidelberg, August 1987. doi:10.1007/3-540-47721-7_12.

[JAC+20] David Jao, Reza Azarderakhsh, Matthew Campagna, Craig Costello, Luca De Feo, Basil Hess, Amir Jalali, Brian Koziel, Brian LaMacchia, Patrick Longa, Michael Naehrig, Joost Renes, Vladimir Soukharev, David Urbanik, Geovandro Pereira, Koray Karabina, and Aaron Hutchinson. SIKE. Technical report, National Institute of Standards and Technology, 2020. available at https://csrc.nist.gov/projects/post-quantum-cryptography/post-quantum-cryptography-standardization/round-3-submissions.

[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*, pages 19–34. Springer, Heidelberg, November / December 2011. doi:10.1007/978-3-642-25405-5_2.

[JMRK23] David Jacquemin, Anisha Mukherjee, Sujoy SINHA ROY, and Péter Kutas. Towards a constant-time implementation of isogeny-based signature, sqisign. Cryptology ePrint Archive, Paper 2023/807, 2023. https://eprint.iacr.org/2023/807. URL: https://eprint.iacr.org/2023/807.

[MMP+23] Luciano Maino, Chloe Martindale, Lorenz Panny, Giacomo Pope, and Benjamin Wesolowski. A direct key recovery attack on SIDH. In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part V*, volume 14008 of *LNCS*, pages 448–471. Springer, Heidelberg, April 2023. doi:10.1007/978-3-031-30589-4_16.

[MN90] François Morain and Jean-Louis Nicolas. On Cornacchia's algorithm for solving the diophantine equation $u^2 + dv^2 = m$, 1990. URL: http://www.lix.polytechnique.fr/Labo/Francois.Morain/Articles/cornac.pdf.

[MR18] Michael Meyer and Steffen Reith. A faster way to the CSIDH. In Debrup Chakraborty and Tetsu Iwata, editors, *INDOCRYPT 2018*, volume 11356 of *LNCS*, pages 137–152. Springer, Heidelberg, December 2018. doi:10.1007/978-3-030-05378-9_8.

[PH78] Stephen Pohlig and Martin Hellman. An improved algorithm for computing logarithms over $\mathrm{GF}(p)$ and its cryptographic significance. *IEEE Transactions on Information Theory*, 24(1):106–110, January 1978. doi:10.1109/TIT.1978.1055817.

[PS00] David Pointcheval and Jacques Stern. Security arguments for digital signatures and blind signatures. *Journal of Cryptology*, 13(3):361–396, June 2000. doi:10.1007/s001450010003.

[PS18] Christophe Petit and Spike Smith. An improvement to the quaternion analogue of the l-isogeny problem. *Presentation at MathCrypt*, 2018.

[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*, pages 229–247. Springer, Heidelberg, 2018. doi:10.1007/978-3-319-79063-3_11.

[Rob23] Damien Robert. Breaking SIDH in polynomial time. In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part V*, volume 14008 of *LNCS*, pages 472–503. Springer, Heidelberg, April 2023. doi:10.1007/978-3-031-30589-4_17.

[SM98] Arne Storjohann and Thom Mulders. Fast algorithms for linear algebra modulo n. In Gianfranco Bilardi, Giuseppe F. Italiano, Andrea Pietracaprina, and Geppino Pucci, editors, *Algorithms — ESA' 98*, pages 139–150, Berlin, Heidelberg, 1998. Springer Berlin Heidelberg. doi:10.1007/3-540-68530-8_12.

[Vél71] Jacques Vélu. Isogénies entre courbes elliptiques. *Comptes Rendus de l'Académie des Sciences de Paris, Séries A*, 273:238–241, 1971.

[Voi21] John Voight. *Quaternion Algebras*. Springer Graduate Texts in Mathematics series, 2021. doi:10.1007/978-3-030-56694-4.

[Wes22] Benjamin Wesolowski. The supersingular isogeny path and endomorphism ring problems are equivalent. In *62nd FOCS*, pages 1100–1111. IEEE Computer Society Press, February 2022. doi:10.1109/FOCS52979.2021.00109.