SQIsign

Algorithm specifications and supporting documentation

Version 2.0

- Marius A. Aardal, *Aarhus University, Denmark*
- Gora Adj, *Technology Innovation Institute, UAE*
- Diego F. Aranha, *Aarhus University, Denmark*
- Andrea Basso, *IBM Research Europe, Switzerland*
- Isaac Andrés Canales Martínez, *Technology Innovation Institute, UAE*
- Jorge Chávez-Saab, *Technology Innovation Institute, UAE*
- Maria Corte-Real Santos, *CNRS* and *ENS de Lyon, France*, previously *University College London, UK*
- Pierrick Dartois, *Bordeaux University Inria Center, France*
- Luca De Feo, *IBM Research Europe, Switzerland*
- Max Duparc, *École Polytechnique Fédérale de Lausanne, Switzerland*
- Jonathan Komada Eriksen, *KU Leuven, Belgium*, previously *NTNU, Norway*
- Tako Boris Fouotsa, *École Polytechnique Fédérale de Lausanne, Switzerland*
- Décio Luiz Gazzoni Filho, *State University of Londrina, Brazil*
- 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*
- Luciano Maino, *University of Bristol, UK*
- Michael Meyer, *University of Regensburg, Germany*
- Kohei Nakagawa, *NTT Social Informatics Laboratories, Japan*
- Hiroshi Onuki, *University of Tokyo, Japan*
- Lorenz Panny, *Technische Universität München, Germany*, previously *Academia Sinica, Taiwan*
- Sikhar Patranabis, *IBM Research India*
- Christophe Petit, *Université libre de Bruxelles, Belgium* and *University of Birmingham, UK*
- Giacomo Pope, *NCC Group, UK* and *University of Bristol, UK*
- Krijn Reijnders, *Radboud University Nijmegen, Netherlands*
- Damien Robert, *Bordeaux University Inria Center, France*
- Francisco Rodríguez Henríquez, *Technology Innovation Institute, UAE*
- Sina Schaeffler, *IBM Research Europe* and *ETH Zürich, Switzerland*
- Benjamin Wesolowski, *CNRS* and *ENS de Lyon, France*

February 5, 2025

contact@sqisign.org

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. SQIsign is conjectured to be secure against quantum computer attacks.

The present specifications of SQIsign start in Chapter 2 and Chapter 3 with an overview of the different mathematical objects used in SQIsign, and a precise description of the algorithms to manipulate them. Then, Chapter 4 details the components of SQIsign itself (key generation, signing, and verification). The following chapters discuss the choice of parameters (Chapter 5), known answer tests (KATs) (Chapter 6), performance (Chapter 7), implementation details (Chapter 8), heuristics and failure cases (Chapter 9), and security (Chapter 10).

1.1. Advantages and limitations

A unique performance profile. The performance profile of SQIsign differs greatly from other signature candidates.

- ⊕ **Very 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. In particular, they are about 7× smaller than Falcon (to be standarized as FN-DSA), and 16-17× smaller than ML-DSA.
- ⊙ **Reasonable efficiency.** While significantly slower than ML-DSA and Falcon, SQIsign has a reasonable efficiency that, arguably, makes it practical for most real-world applications. For example, our 64-bit Intel implementation at NIST security level I has a runtime of about 30 ms for signing and 1.5 ms for verification on an Intel Core i7-13700K (Raptor Lake) x86-64, 3.4 GHz CPU (see Table 2 and Chapter 7). Considerable improvements from earlier versions of SQIsign now place it in the same ballpark as many other post-quantum signature candidates.
- ⊖ **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 difficult to implement, especially for resilience to side-channel attacks.[1] Note that the verification procedure is simpler and significantly faster (see Table 5).

A security assumption from number theory. The security of SQIsign relies on the hardness of a computational problem from number theory: computing the endomorphism ring of a supersingular elliptic curve, *the endomorphism ring problem*.[2] This problem has several advantages.

- ⊕ **Confident tuning of security parameters.** The complexity of the fastest known attacks against SQIsign (i.e., solving the endomorphism ring problem) is well understood. It allows for straightforward, precise estimation of concrete attack costs, hence confident tuning of security parameters. The complexity of the best algorithms for this problem has been very stable.
- ⊕ **Provable security.** The scheme is proven EUF-CMA secure assuming the hardness of a version *with hints* of the endomorphism ring problem.

[1] This is best reflected in the fact that the implementations provided in this submission do not run in constant-time. Nevertheless, as a first step in the right direction, a significant portion of the underlying algorithms for the finite-field, elliptic-curve, pairing, and isogeny arithmetic have been implemented in constant-time.

[2] 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 an easier 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	65	353	148
NIST-III	97	529	224
NIST-V	129	701	292

TABLE 2. SQISIGN performance in 10^6 CPU cycles for the optimized 64-bit Intel implementation on an Intel Core i7-13700K. Results are the median of 1,000 benchmark runs.

Parameter set	Key Gen.	Signing	Verification
NIST-I	43.3	101.6	5.1
NIST-III	134.0	309.2	18.6
NIST-V	212.0	507.5	35.7

⊕ **Random instances are as hard as the worst case.** The endomorphism ring problem enjoys a simple worst-case to average-case self-reduction for the uniform distribution. SQISIGN public keys are at small statistical distance to the uniform distribution, hence the security of the scheme is supported by the *worst-case* endomorphism ring problem.

⊕ **Improving the diversity of security assumptions.** SQISIGN differs greatly from all other signature candidates. The endomorphism ring problem is of a very different nature from the lattice-based assumptions underlying ML-DSA and Falcon, and from all other signature candidates.

⊖ **A relatively new assumption.** On the other hand, the main drawback of this assumption is its relative novelty. Arithmeticians started studying the endomorphism ring problem in 1996 [Koh96], but it has only recently received the spotlight of cryptography.

1.2. 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 (Σ) protocol proves knowledge of an *elliptic curve endomorphism*.

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_{pk}, and their secret is the associated endomorphism ring $\mathrm{End}(E_{\mathsf{pk}})$. With E_{pk} public, the goal of the prover is to convince the verifier that they know $\mathrm{End}(E_{\mathsf{pk}})$. They proceed as follows.

(1) For the commitment phase, the prover generates a random pair $(E_{\mathsf{com}}, \mathrm{End}(E_{\mathsf{com}}))$, and sends E_{com} to the verifier.

(2) For the challenge phase, the verifier generates a random isogeny φ_{chl} from E_{pk} to some other curve E_{chl}, and sends it to the prover.

(3) Given $\mathrm{End}(E_{\mathsf{pk}})$ and $\varphi_{\mathsf{chl}} : E_{\mathsf{pk}} \to E_{\mathsf{chl}}$, the prover can compute $\mathrm{End}(E_{\mathsf{chl}})$. Now, knowing $\mathrm{End}(E_{\mathsf{com}})$ and the freshly computed $\mathrm{End}(E_{\mathsf{chl}})$, the prover can compute an isogeny $\varphi_{\mathsf{rsp}} : E_{\mathsf{com}} \to E_{\mathsf{chl}}$, and send it to the verifier.

(4) The verifier checks that φ_{rsp} is indeed an isogeny from the commitment curve E_{com} to the challenge curve E_{chl}.

The idea is that to compute the response, the prover must use their knowledge of $\mathrm{End}(E_{\mathsf{pk}})$. Intuitively, the protocol asks for the prover to compute isogenies from E_{com} to a somewhat random curve E_{chl}, and that relates to knowledge of $\mathrm{End}(E_{\mathsf{pk}})$ 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_{com} by choosing a random isogeny $\varphi_{\mathrm{com}}^{\mathrm{cheat}} : E_{\mathsf{pk}} \to E_{\mathsf{com}}$. They may not be able to compute $\mathrm{End}(E_{\mathsf{com}})$, but it does not matter. In response to $\varphi_{\mathsf{chl}} : E_{\mathsf{pk}} \to E_{\mathsf{chl}}$, they would simply respond with $\varphi_{\mathrm{rsp}}^{\mathrm{cheat}} = \varphi_{\mathsf{chl}} \circ \widehat{\varphi_{\mathrm{com}}^{\mathrm{cheat}}} : E_{\mathsf{com}} \to E_{\mathsf{chl}}$ (where $\widehat{\varphi_{\mathrm{com}}^{\mathrm{cheat}}} : E_{\mathsf{com}} \to E_{\mathsf{pk}}$ is the *dual* of $\varphi_{\mathrm{com}}^{\mathrm{cheat}}$).

There is a simple fix to this issue, by ensuring that φ_{chl} is not a "sub-isogeny" of φ_{rsp}. With this fix, one can actually prove that this protocol proves knowledge of *at least* some non-trivial part of $\mathrm{End}(E_{\mathsf{pk}})$. It is known that computing a non-trivial part of $\mathrm{End}(E_{\mathsf{pk}})$ is as hard as computing the full $\mathrm{End}(E_{\mathsf{pk}})$ [PW24].

The SQIsign protocol described in detail in Chapter 4 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 $(E_{\mathsf{pk}}, \mathrm{End}(E_{\mathsf{pk}}))$, we think of them as a random isogeny $\varphi_{\mathsf{sk}} : E_0 \to E_{\mathsf{pk}}$. Similarly, we generate the commitment via a random isogeny $\varphi_{\mathrm{com}} : E_0 \to E_{\mathsf{com}}$ instead of a random pair $(E_{\mathsf{com}}, \mathrm{End}(E_{\mathsf{com}}))$. 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 $E_0 \to E$.

1.3. Differences with the round-1 SQIsign submission

The main difference between the *round-1 SQIsign submission* and the present *round-2 SQIsign submission* is the implementation of the improvements described in [BDD+24]. The structure of the scheme, as described in Section 1.2, remains unchanged. The differences amount to the following four points:

(1) **Uniform keys.** The key generation procedure now selects a uniformly random supersingular elliptic curve. This improves the theoretical security guarantee, because the underlying computational problem (the endomorphism ring problem) benefits from a worst-case to average-case self-reduction for the uniform distribution: key recovery is now provably as hard as the hardest instance of the endomorphism ring problem.

(2) **Response sampled from a well-understood distribution.** The response phase, as described in Section 1.2, requires sampling a "response isogeny" $\varphi_{\mathrm{rsp}} : E_{\mathsf{com}} \to E_{\mathsf{chl}}$ from a collection of possible responses. In the previous version of SQIsign, this response was sampled in an *ad hoc* manner, which was hard to analyze (a concern for the *zero-knowledge* property of the scheme), and forced the degree of φ_{rsp} to be very large (causing the scheme to be slow). The response is now sampled from a natural, well-understood distribution: the uniform distribution on the finite set of isogenies $E_{\mathsf{com}} \to E_{\mathsf{chl}}$ of bounded degree. This improves the theoretical security guarantee by removing *ad hoc* assumptions from the zero-knowledge property.

(3) **Response represented by interpolation data.** The response isogeny is now represented by interpolation data: the images of a few points through the isogeny. The previous version of SQIsign used another representation (the isogeny path representation) which only works for special isogenies. The interpolation method allows one to represent any isogeny. This allows one to represent isogenies sampled from the aforementioned uniform distribution. Combined with the aforementioned improvement on the degree of φ_{rsp}, this method results in a significant speedup of SQIsign. This interpolation method requires the computation of isogenies between abelian surfaces (a two-dimensional analog of elliptic curves).

(4) **Much better performance, in all metrics.** As a result of the changes outlined above, together with an improved implementation, the round-2 version of SQIsign drastically outperforms the round-1 version: for security level I, the optimized implementation of signing is now nearly $20\times$ faster, at 103.0 Mcycles, and verification is more than $6\times$ faster, at 5.1 Mcycles. At higher security levels, the improvements are even larger. Beyond the running time improvements, signatures in the round-2 version are also smaller, by about 14%.

Concretely, the material on the KLPT algorithm has been removed (Section *2.5.2. The KLPT algorithm and generalizations* in the previous version), and material on the computation of isogenies in dimension 2 has been added (Section 2.4 and Section 8.5). Modifications throughout the document reflect this new approach.

1.4. Notation

Assignment of a certain value α to a variable x is denoted by $x \leftarrow \alpha$. Algorithms are typeset in sans serif font, e.g., IdealToIsogeny. We use O for big-O notation, and $f = \Theta(g)$ whenever f is bounded from below and from above by g asymptotically. We denote composition of functions using $\circ$, e.g., $(f \circ g)(x) = f(g(x))$.

We denote matrices with bold capital letters, e.g., $\mathbf{M}, \mathbf{A}$, and vectors with bold small letters, e.g., $\mathbf{v}, \mathbf{x}$. We denote the transpose of a matrix $\mathbf{M}$ by $\mathbf{M}^t$. Elliptic curves and abelian surfaces are usually denoted with capital letters, e.g., E, E', A, and their points similarly, e.g., P, Q.

We denote by $\mathbf{M} \cdot P$ the application of a matrix $\mathbf{M}$ to a point P on an elliptic curve or abelian surface (in any coordinate system). Here, we view P as a column vector, and multiplication $\cdot$ is performed as the usual matrix-vector multiplication.

Isogenies between elliptic curves are denoted with Greek letters, e.g., φ, ψ, τ; endomorphisms often use θ, and higher-dimensional isogenies use capital Greek letters, e.g., Φ, Ψ. Quaternions are usually denoted with Greek letters too, e.g., α, β, whereas quaternion ideals are denoted by capital Roman letters, e.g., I, J. The generators of the quaternion algebra are denoted by i, j, k. More specific notation used in this document is shown in Table 3.

There is an unfortunate clash with the usual indices i, j, k used for enumeration in **for** loops and the generators of the quaternion algebra. We use the following convention: for algorithms that do not involve quaternion algebras, the symbols i, j, k lose their usual definition as generators and should be viewed simply as indices. Otherwise, we use different (less standard) indices. In IdealGenerator, enumeration indices a, b, c, d are used, which clashes with the notation used for theta null points in Section 8.5. Here, the same convention applies.

1.4.1. Exception handling

For clarity of exposition, throughout this specification, we use exceptions for error handling. We use the following convention.

```
1: try
2:     Do something that might go wrong.
3:     if Something is actually wrong then
4:         raise Exception("Description of the error")      // Exit the try environment and go to the except environment
5: except
6:     Try to recover or show the user an error message.
```

Additionally, we use the **continue** statement to skip the remaining set of instructions inside a loop.

TABLE 3. Notation and parameters of the SQISIGN scheme.

	Scheme Parameters
λ	Security parameter, one of 128, 192 or 256.
p	Size of the base field, a prime of the form $c \cdot 2^f - 1$.
f	The 2-valuation of $p+1$, roughly 2λ.
	Elementary Objects
$\mathbb{Z}/d\mathbb{Z}$	Ring of integers modulo d for some integer d.
$\mathbb{F}_p$	Finite fields with p elements.
$\mathbb{F}_q$	Finite fields with $q = p^k$ elements.
$\mathbb{F}_{p^2}$	Finite fields with p^2 elements.
$\overline{\mathbb{F}}_p$	Algebraic closure of $\mathbb{F}_p$.
$\mathrm{GL_n}(q)$	Group of invertible matrices of size n and elements in $\mathbb{F}_q$.
	Elliptic Curve Objects
E	An elliptic curve over some field $\mathbb{F}_q$.
$E \times E'$	A product of two elliptic curves E and E'.
A	An abelian variety over some field $\mathbb{F}_q$.
$\mathrm{Jac}(C)$	The Jacobian of a hyperelliptic curve C.
$E_{A,B}$	An elliptic curve in Montgomery form with parameters $A, B \in \mathbb{F}_q$.
0_E	The point at infinity of an elliptic curve or abelian variety E.
$E(\mathbb{F}_q)$	The points on E with coordinates in $\mathbb{F}_q$.
$E[n]$	The n-torsion on E.
hint_i	Hints to generate a basis for $E[2^f]$.
$j(E)$	The j-invariant of the elliptic curve E.
x_P and y_P	The x- resp. y-coordinate of a point $P \in E$.
$\varphi : E \to E'$	An isogeny between elliptic curves.
$\Phi : A \to A'$	A higher-dimensional isogeny.
$\mathrm{End}(E)$	The endomorphism ring of an elliptic curve E.
	Quaternion Objects
$B_{p,\infty}$	A quaternion algebra, ramified at p and ∞.
$B_{p,\infty}^\star$	The space of linear functions $B_{p,\infty} \to \mathbb{Q}$.
$\mathcal{O}$	An order in $B_{p,\infty}$.
$\mathcal{O}_L(I)$, $\mathcal{O}_R(I)$	The left, resp. right, order of an ideal $I \subset \mathcal{O}$.
$\mathrm{tr}(\alpha)$	The trace of an element $\alpha \in B_{p,\infty}$.
$\mathrm{nrd}(\alpha)$	The norm of an element $\alpha \in B_{p,\infty}$.
$\mathrm{nrd}(I)$	The norm of an ideal $I \subset \mathcal{O}$.
$[I]^*$	The pullback by an ideal I.
$[I]_*$	The pushforward by an ideal I.
	Protocol Objects
pk	The public key, containing a curve E_{pk} and basis hints $\mathsf{hint}_{\mathsf{pk}}$.
sk	The secret key, containing the curve E_{pk} and $\mathsf{hint}_{\mathsf{pk}}$, the secret ideal I_{sk} and a change-of-basis matrix $\mathbf{M}_{\mathsf{sk}}$.
msg	The message to be signed.
HASH	A hash function, obtained by repeated application of SHAKE256.
com	Commitment, containing a description of the curve E_{com}.
chl	Challenge, an integer between 0 and $2^{e_{\mathsf{chl}}}$ generated by hashing the public key pk, E_{com} and the message msg.
resp	Response, containing a description of an isogeny $E_{\mathsf{com}} \to E_{\mathsf{chl}}$.
E_{aux}	The auxiliary curve required for the higher-dimensional response isogeny.
n_{bt}	The number of backtracking steps in the computation of the response isogeny.
r_{rsp}	The dyadic valuation of $\deg \varphi_{\text{rsp}}$.
σ	Signature, containing E_{aux} together with n_{bt}, r_{rsp}, the change-of-basis matrix $\mathbf{M}_{\mathsf{chl}}$, the challenge chl, $\mathsf{hint}_{\mathsf{aux}}$ and $\mathsf{hint}_{\mathsf{chl}}$.
D_{mix}	Degree of the secret isogeny and commitment isogeny, chosen as the smallest prime larger than $2^{4\lambda}$.
D_{rsp}	Degree of the response isogeny; concretely $D_{\text{rsp}} = 2^{e_{\text{rsp}}}$ where $e_{\text{rsp}} = \lceil \log_2(\sqrt{p}) \rceil$.
D_{chl}	Degree of the challenge isogeny, defined as $D_{\text{chl}} = 2^f$.
e_{chl}	Size of the challenge space, given by $e_{\text{chl}} = f - e_{\text{rsp}}$.

CHAPTER 2

Basics of Isogenies

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. This chapter describes the first world, which requires working finite fields (Section 2.1), elliptic curves (Section 2.2) and isogenies, both one-dimensional (Section 2.3) and higher-dimensional (Section 2.4). This is as much as is needed for verification. The world of quaternions is described in Chapter 3, and specifically how these two worlds connect in Section 3.2.

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 $c = c_0 + c_1 \cdot i$ 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 $c = c_0 + c_1 \cdot i$ with $c_0 = a_0b_0 - a_1b_1$ and $c_1 = a_0b_1 + a_1b_0$, 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 $a^{-1} = (a_0N^{-1}) + (-a_1N^{-1}) \cdot i$, 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 $\mathbb{F}_{p^2}$. We use [AAA+25] to define a canonical square root of $a = a_0 + a_1 \cdot i$ as

$$\sqrt{a} = (-i)^{\frac{1-\chi}{2}} S \cdot (a + \delta), \text{ where } \delta = \sqrt{a_0^2 + a_1^2}, \quad S = [2(a_0 + \delta)]^{\frac{p-3}{4}} \quad \text{and} \quad \chi = [2(a_0 + \delta)]^{\frac{p-1}{2}}. \tag{2}$$

Algorithm 8.2 gives a way to efficiently compute this square root.

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 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 0_E. 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 $\mathbb{F}_q$, where A is often called the *Montgomery coefficient*.

Two Montgomery curves are said to be *isomorphic* over $\mathbb{F}_q$ if there is a linear change of coordinates $(x, y) \mapsto (Dx + R, Cy)$, 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 0_E). 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.

2.2.1.1. Isomorphisms between Montgomery curves. 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. Given two curves $E_{A,B}$ and $E_{A',B'}$ that are isomorphic, the change of coordinates $(x, y) \mapsto (Dx + R, Cy)$ that maps the former onto the latter is given by

$$D = \lambda^2\frac{B'}{B}, \qquad R = \lambda^2\frac{AB'}{3B} - \frac{A'}{3}, \qquad C = \frac{\lambda^3 B'}{B}, \qquad \lambda := \sqrt{\frac{B(2A'^3 - 9A')(3-A^2)}{B'(2A^3-9A)(3-A'^2)}}. \tag{6}$$

These formulas were obtained by first converting the curves to Short Weierstrass models $E_{a,b}, E_{a',b'}$ using the formula from [CS18, Section 2.4], and then using the isomorphism $E_{a,b} \to E_{a',b'}$ given by $(x, y) \mapsto (\lambda^2 x, \lambda^3 y)$ with $\lambda^2 = (a'b)/(ab')$. The formulas are implemented by the IsomorphismMontgomeryCurves algorithm described in Section 8.2.2.

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 0_E is the identity element.

In what follows, for a point $P \neq 0_E$, 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]). Furthermore, we mostly use x-only arithmetic and represent points only as $P = (X_P : Z_P)$, which means that points are only defined up to the sign of their y-coordinates.

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 $P + 0_E = 0_E + P = P$.

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 $[2]P = 0_E$.

2.2.2.3. Scalar multiplication. Using the abelian group law, 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 $[0]P = 0_E$.

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 $[m]P = 0_E$ the order of P.

2.2.2.4. Point difference. Given the x-coordinates x_P and x_Q of two points $P, Q \in E(\mathbb{F}_{p^2})$, we can deterministically compute the set $\{r_-, r_+\} = \{x_{P-Q}, x_{P+Q}\}$ using the following formula from [RS17, Prop. 3]:

$$r_\pm = \frac{B_{XZ} \pm \sqrt{B_{XZ}^2 - B_{ZZ}B_{XX}}}{B_{ZZ}}, \tag{7}$$

where

$$\begin{aligned} B_{XX} &= (x_P x_Q - 1)^2, \\ B_{XZ} &= (x_P x_Q + 1)(x_P + x_Q) + 2A x_P x_Q, \\ B_{ZZ} &= (x_P - x_Q)^2. \end{aligned}$$

As $x_Q = x_{-Q}$, we cannot determine which of r_+, r_- is x_{P-Q} and which is x_{P+Q} based on x coordinates alone. However, in practice we are also free to replace Q by $-Q$ and so it suffices to deterministically chose one of the two. We make this choice by always using the formula for r_+, with the choice of square root described in Section 2.1.2. This routine is referred to as ProjectiveDifference, since in practice it is implemented in projective coordinates as will be discussed in Section 8.2.3.

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 $[m]P = 0_E$. 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 $E[m]$. The reference implementation represents 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 $(-R, -S)$. Furthermore, this requires us to use Ladder (see Section 8.2.1) to compute the x-coordinate of $[n]R$ given only x_R.

Throughout this document, f denotes the largest integer such that 2^f divides $p+1$, for p the base prime of the finite field. In SQIsign, we need to generate a basis (R, S) for $E[2^e]$ for $e \leq f$, which is well-studied [CJL+17; SEMR24; ZSP+18]. We describe the deterministic generation of a basis (R, Q) for $E[2^e]$ in TorsionBasisToHint, using a modified version of [ZSP+18, Alg. 3.1]: This algorithm computes the quadratic residuosity of A to ensure that $x_R \leftarrow -A/(1 + i \cdot b)$ is a non-square in $\mathbb{F}_{p^2}$, for a square b. Then, if $x_R \in E(\mathbb{F}_{p^2})$, this ensures that $x_S = -x_R - A$ is also a non-square, and that together (R, S) is a basis of $E[2^e]$, when multiplied by the cofactor $(p+1)/2^e$. We adjust this algorithm by using that $b \in \mathbb{F}_p$ is always a square in $\mathbb{F}_{p^2}$, hence when A is non-square, any $\mathbb{F}_p$-multiple $b \cdot A$ is of the right form for x_R. One can show that for a basis (R, S) sampled using this algorithm, the point $R + S$ is always above $(0, 0)$, which is a property we assume in later algorithms. To explicitly have a basis point above $(0, 0)$, we therefore use $(R, R+S)$ as our basis, which we obtain from permuting (x_R, x_S, x_{RS}) to (x_R, x_{RS}, x_S). Computing an x_{RS} given x_R and x_S must be done deterministically, however, it is impossible to know given only x_R and x_S if x_{RS} is the x-coordinate of $R + S$ or $R - S$. This is not an issue, as long as an implementation makes the same deterministic choice as the reference implementation, which uses ProjectiveDifference (see Section 8.2.3).

Sampling such a torsion basis for $E_A[2^f]$ can be sped up for the verifier: the algorithm requires knowledge of the quadratic residuosity of A and the correct index used to find x_P. Both of these can be provided in the signature as *hints*: the signer uses TorsionBasisToHint, which generates a basis and the corresponding hints, and includes the hints in the signature. The verifier uses TorsionBasisFromHint, which takes the hints as input and returns the basis (x_R, x_S, x_{RS}).

Note that a basis returned using TorsionBasisFromHint is guaranteed to have two values x_R and x_S on the same twist, even when these values are not verified as points on E. This causes no problems, as long as the order of these points are verified whenever they are used, which implicitly ensures they lie on E. The reference implementation does this as described in Section 8.4 and Section 8.5.

For the particular elliptic curve E_0, TorsionBasis (Algorithm 2.3) will be used to generate a deterministic basis of $E_0[2^f]$. This algorithm is only used for parameter generation, so does not require careful optimization.

2.2.4. Discrete logarithms

Given a finite-field element $\zeta \in \mathbb{F}_{p^2}$ and a power ζ^k for an unknown $k \in \mathbb{Z}$, the discrete logarithm problem (DLP) asks to recover k. For elements of large prime order, this problem is suspected to be hard for classical

Algorithm 2.1 TorsionBasisToHint(A, e)

Input: A non-zero affine Montgomery coefficient A and an integer $e \leq f$.
Output: An x-only basis (x_R, x_S, x_{RS}) of $E[2^e]$, together with two hints h_A, h.

if A is a square **then**
  $h_A \leftarrow 1$
else
  $h_A \leftarrow 0$
$h \leftarrow 0$
if A is square **then**
  repeat $h \leftarrow h + 1,\ x_R \leftarrow -1/(1 + i \cdot h) \cdot A$
  until $(1 + h^2)$ is not a square **and** $x_R \in E_A(\mathbb{F}_{p^2})$
else
  repeat $h \leftarrow h + 1,\ x_R \leftarrow h \cdot A$
  until $x_R \in E_A(\mathbb{F}_{p^2})$
$x_{RS} \leftarrow -x_R - A$
$x_R \leftarrow \mathsf{Ladder}((x_R : 1), (A + 2 : 4), \frac{p+1}{2^e})$
$x_{RS} \leftarrow \mathsf{Ladder}((x_{RS} : 1), (A + 2 : 4), \frac{p+1}{2^e})$
$x_S \leftarrow \mathsf{ProjectiveDifference}(x_R, x_{RS}, (A : 1))$
if $h \geq 128$ **then**
  $h \leftarrow 0$
return (x_R, x_S, x_{RS}) and (h_A, h)

Algorithm 2.2 TorsionBasisFromHint(E, h)

Input: A non-zero affine Montgomery coefficient A, an integer $e < f$, together with two hints h_A, h.
Output: An x-only basis (x_R, x_S, x_{RS}) of $E[2^e]$.

if $h = 0$ **then**
  $(x_R, x_S, x_{RS}), (h_A, h) \leftarrow \mathsf{TorsionBasisToHint}(A, e)$
  return (x_R, x_S, x_{RS})
else
  if $h_A = 1$ **then**
    $x_R \leftarrow -1/(1 + i \cdot h) \cdot A$
  else
    $x_R \leftarrow h \cdot A$
  $x_{RS} \leftarrow -x_R - A$
  $x_R \leftarrow \mathsf{Ladder}((x_R : 1), (A + 2 : 4), \frac{p+1}{2^e})$
  $x_{RS} \leftarrow \mathsf{Ladder}((x_{RS} : 1), (A + 2 : 4), \frac{p+1}{2^e})$
  $x_S \leftarrow \mathsf{ProjectiveDifference}(x_R, x_{RS}, (A : 1))$
  return (x_R, x_S, x_{RS})

computers, and underlies the security of traditional Diffie–Hellman cryptography. However, when the order of ζ is smooth enough, this problem can be efficiently solved. Let μ_n denote the group of n-th roots of unity, that is, $\mu_n := \{\zeta \in \overline{\mathbb{F}}_p \mid \zeta^n = 1\}$. SQIsign only uses primes p such that $\mu_{2^f} \subset \mathbb{F}_{p^2}^*$, where f is the largest integer such that 2^f divides $p + 1$. In particular, in SQIsign we consider elements $\zeta \in \mu_{2^e} \subset \mathbb{F}_{p^2}^*$ with $e \leq f$, i.e., elements whose multiplicative order divides 2^e in $\mathbb{F}_{p^2}$.

Several algorithms exist to solve such discrete logarithms in μ_{2^e}, all tracing back to Pohlig–Hellman [PH78]. In our implementation, we use the iterative algorithm NormalizedDlog, shown in Algorithm 2.4, to compute this discrete logarithm between two elements using the Pohlig–Hellman algorithm. SQIsign only applies this algorithm to the output of pairing computations (see Section 2.2.5).

Algorithm 2.3 $\mathsf{TorsionBasis}_0(\mathbb{F}_{p^2})$

Input: The field $\mathbb{F}_{p^2} = \mathbb{F}_p(i)$.
Output: A basis (P, Q) of $E_0[2^f]$ such that $[2^{f-1}]Q = (0, 0)$.

$a \leftarrow 0$, $P \leftarrow 0_E$
repeat
  $a \leftarrow a + 1$
  $x_P \leftarrow 1 + ia$
  $y_P \leftarrow (x_P^3 + x_P)^{\frac{p+1}{4}}$
  if $y_P^2 = x_P^3 + x_P$ **then**
    $P \leftarrow \left[\frac{p+1}{2^f}\right](x_P, y_P)$
until $[2^{f-1}]P \neq 0_E$ and $[2^f]P = 0_E$
$b \leftarrow a$, $Q \leftarrow 0_E$
repeat
  $b \leftarrow b + 1$
  $x_Q \leftarrow 1 + ib$
  $y_Q \leftarrow (x_Q^3 + x_Q)^{\frac{p+1}{4}}$
  if $y_Q^2 = x_Q^3 + x_Q$ **then**
    $Q \leftarrow \left[\frac{p+1}{2^f}\right](x_Q, y_Q)$
until $[2^{f-1}]Q \notin \{0_E, [2^{f-1}]P\}$ and $[2^f]Q = 0_E$
if $[2^{f-1}]P = (0, 0)$ **then**
  $(P, Q) \leftarrow (Q, P)$
else if $[2^{f-1}](P + Q) = (0, 0)$ **then**
  $Q \leftarrow P + Q$
return (P, Q)

Algorithm 2.4 $\mathsf{NormalizedDlog}(\zeta_0, \zeta_1)$

Input: Two values $\zeta_0, \zeta_1 \in \mu_{2^e}$ with $e \leq f$.
Output: The value $k \in [0..2^e]$ such that $\zeta_1 = \zeta_0^k$.

if $e = 1$ **then**
  return k such that $\zeta_1 = \zeta_0^k$ by exhaustive search
$e' \leftarrow \lfloor e/2 \rfloor$
$\zeta_0' \leftarrow \zeta_0^{2^{e-e'}}$
$\zeta_1' \leftarrow \zeta_1^{2^{e-e'}}$
$k' \leftarrow \mathsf{NormalizedDlog}(\zeta_0', \zeta_1')$
$\zeta_0'' \leftarrow \zeta_0^{2^{e'}}$
$\zeta_1'' \leftarrow \zeta_1^{2^{e'}}$
$k'' \leftarrow \mathsf{NormalizedDlog}(\zeta_0'', \zeta_1'')$
return $k = k' + 2^{e'} k''$

2.2.5. Pairings

This section introduces the Weil and Tate-Lichtenbaum pairings of level n. One approach to efficiently compute the discrete logarithm between points is through pairings. This has been extensively explored in other isogeny-based schemes in the past, such as SIKE, and has also been proposed to achieve moderate acceleration in the first round of SQIsign [LWXZ24]. Pairings allow us to translate the elliptic-curve discrete logarithm problem into a finite-field discrete logarithm problem, which can then be solved efficiently with a function like NormalizedDlog. This approach is particularly attractive in the context of SQIsign, as we are primarily concerned with discrete logarithm computations of points belonging to $E[2^e]$, where $e \leq f$ and 2^f is the largest power of 2 dividing $p + 1$. It follows that pairings of degree 2^f have an embedding degree of 2, which is very helpful for obtaining faster pairing computations.

2.2.5.1. Pairings on elliptic curves. Pairings are bilinear maps $A \times B \to C$ between abelian groups A, B, and C. Most relevant for cryptography are the Weil and Tate-Lichtenbaum pairing of level n, where A and B are subgroups or quotient groups of $E(\mathbb{F}_q)$ and C is the group of n-th roots of unity $\mu_n \subseteq E(\mathbb{F}_q)$. Note that SQIsign only requires pairings on elliptic curves, although one can similarly compute such pairings on abelian varieties.

The Weil pairing. Let E be an elliptic curve over $\mathbb{F}_q$ and let $n \in \mathbb{N}$ coprime to $\operatorname{char} \mathbb{F}_q$. Then the *Weil pairing* of level n, introduced by Weil [Wei40], is a map

$$e_n : E[n] \times E[n] \to \mu_n,$$

which is bilinear, alternating, non-degenerate, and Galois invariant. The Weil pairing can also be defined for higher-dimensional abelian varieties (see Section 2.4), where this pairing is used implicitly. The reference implementation, therefore, does not concretely compute Weil pairings.

The Tate-Lichtenbaum pairing. The *Tate-Lichtenbaum pairing* is a pairing defined by Tate [Tat62] for abelian varieties over local fields, with Lichtenbaum [Lic69] showing its efficient computation for Jacobians of curves. Frey and Rück [FR94] show its cryptographic relevance, including an efficient computation over finite fields. Assume $\mu_n \subset \mathbb{F}_q$. We define the *unreduced* Tate-Lichtenbaum pairing of level n as a map

$$T_n : E(\mathbb{F}_q)[n] \times E(\mathbb{F}_q)/nE(\mathbb{F}_q) \to \mathbb{F}_q^* / (\mathbb{F}_q^*)^n.$$

As a result, $T_n(P, Q)$ is unique up to n-th powers. In a cryptographic context, we prefer a well-defined, unique value which we can achieve by raising the result to the power $(q-1)/n$. This ensures the final result is a unique value in μ_n. Thus, we get the *reduced* Tate-Lichtenbaum pairing of level n,

$$t_n : E(\mathbb{F}_q)[n] \times E(\mathbb{F}_q)/nE(\mathbb{F}_q) \to \mu_n, \qquad (P, Q) \mapsto T_n(P, Q)^{(q-1)/n}.$$

Both the reduced and unreduced Tate-Lichtenbaum pairing are bilinear, non-degenerate and Galois invariant. When $E/\mathbb{F}_{p^2}$ is a maximal supersingular curve, and $n | p+1$, then the reduced Tate-Lichtenbaum pairing is alternate.

2.2.5.2. Use of pairings in SQIsign. The Tate pairing t_n allows us to solve discrete logarithm problems faster. In the following, we will use the fact that for $P \in E[n]$, the Tate-Lichtenbaum of $t_n(P, P)$ vanishes, i.e., $t_n(P, P) = 1$. For example, if $E[n]$ has a basis $P_1, P_2 \in E(\mathbb{F}_q)$ with $\zeta = t_n(P_1, P_2)$ an n-th primitive root of unity, and we have $Q \in E[n]$ which we want to express in this basis as $Q = [a_1]P_1 + [a_2]P_2$, then we find that

$$t_n(P_1, Q) = t_n(P_1, [a_1]P_1 + [a_2]P_2) = t_n(P_1, P_1)^{a_1} \cdot t_n(P_1, P_2)^{a_2} = t_n(P_1, P_2)^{a_2} = \zeta^{a_2}.$$

Similarly, $t_n(P_2, Q) = t_n(P_2, P_1)^{a_1} = \zeta^{-a_1}$, hence, given ζ, we can compute two Tate pairings of level n and solve for a_1, a_2 by solving two discrete logarithms in μ_n using NormalizedDlog. In SQIsign, we use Tate pairings in such a manner for the case $n = 2^e$ for $e \leq f$ to compute the matrix associated to a change of basis: If (P_1, P_2) is a basis for $E[2^f]$, which ensures $\zeta = t_{2^f}(P_1, P_2)$ is a primitive 2^f-th root of unity, and (Q_1, Q_2) is a basis for $E[2^e]$, then the Tate pairing allows us to compute x_1, x_2, x_3, x_4 such that

$$\begin{pmatrix} x_1 & x_2 \\ x_3 & x_4 \end{pmatrix} \cdot \begin{pmatrix} P_1 \\ P_2 \end{pmatrix} = \begin{pmatrix} Q_1 \\ Q_2 \end{pmatrix}.$$

We describe this more precisely in ChangeOfBasis (Algorithm 2.5).[1] Furthermore, if we want to compute x_i that change a basis (Q_1, Q_2) of $E[2^e]$ into the basis $([2^{f-e}]P_1, [2^{f-e}]P_2)$, we can apply the above algorithm and invert the resulting matrix.

There are several methods to compute t_n efficiently; see Section 8.3 for the choice made in SQIsign and details on these algorithms.

Algorithm 2.5 $\mathsf{ChangeOfBasis}_{2^e}(E, (P_1, P_2), (Q_1, Q_2))$

Input: A basis (P_1, P_2) for $E[2^f]$ and a basis (Q_1, Q_2) for $E[2^e]$.
Output: A change-of-basis matrix (x_i), with $1 \leq i \leq 4$, so that $Q_1 = [x_1]P_1 + [x_2]P_2$ and $Q_2 = [x_3]P_1 + [x_4]P_2$.

1: $\zeta \leftarrow t_{2^e}(P_1, P_2)$
2: $\zeta_1 \leftarrow t_{2^e}(Q_1, P_2),\ \zeta_2 \leftarrow 1/t_{2^e}(Q_1, P_1),\ \zeta_3 \leftarrow t_{2^e}(Q_2, P_2),\ \zeta_4 \leftarrow 1/t_{2^e}(Q_2, P_1)$
3: **for** i **from** 1 **up to** 4 **do**
4: $\quad x_i \leftarrow 2^{f-e} \cdot \log_\zeta(\zeta_i)$
5: **return** (x_1, x_2, x_3, x_4)

[1] One may also use the Weil pairing to compute the change of basis, although at an increase in cost.

2.3. Isogenies between elliptic curves

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 $\varphi(0_{E_1}) = 0_{E_2}$. 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 $\ker(\varphi) = \{P \in E_1 \mid \varphi(P) = 0_{E_2}\}$. 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 $\tilde{O}(\sqrt{N})$, 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, etc. In particular, we compute φ through $\varphi = \varphi_E \circ \cdots \circ \varphi_2 \circ \varphi_1$ where $E = \sum e_i$. 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 degree 2 and 4. 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 here, our implementation uses projective coordinates; see Section 8.4 for more details.

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:

- $Q = (0, 0)$: 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 [JAC+20; Ren18] 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.

In Chapter 8, we depict algorithms implementing the case $Q = (x_Q, 0)$ in TwoIsogenyCodomain (which, given Q, computes the codomain of the 2-isogeny) and TwoIsogenyEval (which, given Q and another point P, computes the image of P under the isogeny generated by Q). The singular case $Q = (0, 0)$ is dealt with analogously in TwoIsogenyCodomainSingular and TwoIsogenyEvalSingular.

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.

In Chapter 8, we depict algorithms implementing the case $Q = (x_Q, y_Q)$ for $x_Q \neq \pm 1$ in FourIsogenyCodomain (which, given Q, computes the codomain of the 4-isogeny) and FourIsogenyEval (which, given Q and another point P, computes the image of P under the isogeny generated by Q). We do not require the other cases in SQISIGN.

2.4. Isogenies between principally polarized abelian surfaces

In SQISIGN, we use isogenies in dimension 2 as a tool to efficiently compute isogenies between elliptic curves of non-smooth degree. In this section, we introduce the necessary background for their computation.

Principally polarized abelian surfaces (PPAS) are a natural generalization of elliptic curves (see Section 2.2) to two dimensions. In particular, PPAS's are geometric objects defined by polynomial equations to which we can associate a group whose group law is given by rational functions (i.e., fractions of polynomials). PPAS are isomorphic (over an algebraically closed field) to either one of the following [OU73]:

(1) A product of elliptic curves $E_1 \times E_2$;
(2) A Jacobian $\mathrm{Jac}(C)$ of a *genus-2 hyperelliptic curve* C.

The arithmetic on the Cartesian product $A := E_1 \times E_2$ of elliptic curves E_1, E_2 follows immediately from the arithmetic of elliptic curves. Indeed, the addition of two points $(P_1, P_2), (Q_1, Q_2) \in A$ is defined as

$$(P_1, P_2) \oplus (Q_1, Q_2) := (P_1 \oplus_1 Q_1, P_2 \oplus_2 Q_2),$$

where $\oplus_1$ and $\oplus_2$ is the group law on the elliptic curves E_1 and E_2, respectively. The group law $\oplus$ has identity $0_A := (0_{E_1}, 0_{E_2})$.

We briefly detail the second type of PPAS. Every hyperelliptic curve of genus-2 defined over $\mathbb{F}_{p^2}$ can be written in the form

$$C\colon y^2 = f(x), \text{ where } f \text{ squarefree with } \deg(f) = 5 \text{ or } 6,$$

when $p > 5$. Unlike elliptic curves, the curve C does not form a group under point addition. Instead, we construct the *Jacobian* $\mathrm{Jac}(C)$ [Mil86], which is the group associated to the curve C. Note that in the case of an elliptic

curve E we have $\mathrm{Jac}(E) \cong E$, which is why the construction of the Jacobian can usually be avoided. The group law on $A := \mathrm{Jac}(C)$ with identity 0_A can be computed using Cantor's algorithms [Can87].

An isogeny $\Phi : A_1 \longrightarrow A_2$ between PPAS is a surjective map defined coordinate-wise by rational fractions which is also a group homomorphism. There are three types of isogenies Φ we work with, depending on what type of PPAS A_1 and A_2:

- If $A_1 = E_1 \times E_2$ and $A_2 = \mathrm{Jac}(C)$ for a genus-2 hyperelliptic curve C, then Φ is a *gluing isogeny*.
- If $A_2 = E_1 \times E_2$ and $A_1 = \mathrm{Jac}(C)$, then Φ is a *splitting isogeny*.
- Otherwise, A_1 and A_2 are both Jacobians and Φ is called *generic*.

Similarly to elliptic curve isogenies, isogenies between PPAS have a finite kernel $\ker(\Phi) = \{P \in A_1 \mid \Phi(P) = 0_{A_2}\}$ and are determined by their kernel up to post composition with an isomorphism. We say that Φ is an (N, N)-isogeny when its kernel $\ker(\Phi)$ is generated by two linearly independent points $P, Q \in A_1$ of order N. In this context, linearly independent means that for all integers k and l, we have $[k]P + [l]Q = 0_{A_1}$ if and only if N divides k and l. We write $\ker(\Phi) = \langle P \rangle \oplus \langle Q \rangle$. The isogeny Φ can then be represented by P and Q and computed from these points with generalizations of Vélu's formulas in time $O(N^2)$ [LR22]. Not all choices of linearly independent N-torsion points P, Q define an isogeny. To define an isogeny of PPAS, it is necessary and sufficient that P and Q are *isotropic*, i.e., P, Q have trivial N-Weil pairing:[2] $e_N(P, Q) = 1$.

Suppose N has prime decomposition $\prod \ell_i^{e_i}$. Rather than computing an (N, N)-isogeny Φ with complexity $O(N^2)$, we decompose this isogeny as

$$\Phi = \Phi_r \circ \cdots \circ \Phi_1,$$

where the Φ_i are (ℓ_i, ℓ_i)-isogenies. The isogeny Φ can now be computed more efficiently in $O(\sum e_i \ell_i^2)$. For SQIsign, we specialize to the case of $(2^e, 2^e)$-isogenies

$$\Phi : E_1 \times E_2 \longrightarrow E_3 \times E_4$$

between elliptic curve products $E_1 \times E_2$ and $E_3 \times E_4$ defined over $\mathbb{F}_{p^2}$, that we decompose into a chain of $(2, 2)$-isogenies of length e. For cryptographic sized primes p, we expect the first isogeny of the chain $\Phi_1 : E_1 \times E_2 \longrightarrow A_1$ to be a gluing isogeny, the last step $\Phi_e : A_{e-1} \longrightarrow E_3 \times E_4$ to be a splitting isogeny, and all intermediate isogenies to be generic.

2.4.0.1. Implementation details for $(2, 2)$-isogenies. In Section 8.5, we give algorithmic details on how $(2, 2)$-isogenies are computed. For practical purposes, we use *theta coordinates of level* 2 to represent points on a PPAS. This is analogous to the use of the x-coordinate on elliptic curves. In particular, we highlight the following algorithms that are depicted:

- ThetaDBL: on input of a point P (in theta coordinates) and the constants `consts`, outputs $[2]P$ (in theta coordinates).
- GenericCodomainWith8Torsion: given theta coordinates of 8-torsion points T_1'', T_2'' on domain surface A, this algorithm outputs the dual theta null point $(\alpha : \beta : \gamma : \delta)$, its inverse $(\alpha^{-1} : \beta^{-1} : \gamma^{-1} : \delta^{-1})$, and the theta null point 0_B of the image B of the isogeny $\Phi : A \to B$ with $\ker(\Phi) = \langle [4]T_1'' \rangle \oplus \langle [4]T_2'' \rangle$.
- GenericCodomainWith4Torsion: given theta coordinates of a 4-torsion point T_1' on domain surface A satisfying $[2]T_1' \in \ker(\Phi)$, and the theta null point 0_A of A, this algorithm outputs the dual theta null point $(\alpha : \beta : \gamma : \delta)$, its inverse $(\alpha^{-1} : \beta^{-1} : \gamma^{-1} : \delta^{-1})$, and the theta null point 0_B of the image B of $\Phi : A \to B$.
- GenericCodomain: given the theta null point 0_A of domain A, computes the dual theta null point $(\alpha : \beta : \gamma : \delta)$, its inverse $(\alpha^{-1} : \beta^{-1} : \gamma^{-1} : \delta^{-1})$, and the theta null point 0_B of the image B of $(2, 2)$-isogeny $\Phi : A \to B$.
- GenericEval: given a point P (in theta coordinates) on domain surface A and the point $(\alpha^{-1} : \beta^{-1} : \gamma^{-1} : \delta^{-1})$, outputs the theta coordinates of $\Phi(P)$.
- GluingCodomain: given theta coordinates of 8-torsion points T_1'', T_2'' on domain product surface, this algorithm outputs the dual theta null point $(\alpha : \beta : \gamma : 0)$, its "inverse" $(\alpha^{-1} : \beta^{-1} : \gamma^{-1} : 0)$, the theta null point 0_B of the image surface A of the isogeny $\Phi : E_1 \times E_2 \to A$ with $\ker(\Phi) = \langle [4]T_1'' \rangle \oplus \langle [4]T_2'' \rangle$, J the dual of the theta point $\Phi(T_1'')$ and a change-of-basis matrix $\mathbf{N}$ (re-used for evaluation).
- GluingEval: given a point $P \in E_1 \times E_2$, T_1'' a 8-torsion point such that $[4]T_1'' \in \ker(\Phi)$, J the dual of the theta point $\Phi(T_1'')$ on A, and the change-of-basis matrix $\mathbf{N}$ computed during GluingCodomain, outputs the theta coordinates of $\Phi(P)$.

[2] The N-Weil pairing can be defined on PPAS as is done with elliptic curves.

- GluingEvalSpecial: given a point $P \in E_1 \times E_2$ of the form $(P_1, 0)$ or $(0, P_2)$, $(\alpha^{-1} : \beta^{-1} : \gamma^{-1} : 0)$ the "inverse" of the dual theta null point over A, and the change-of-basis matrix $\mathbf{N}$ computed during GluingCodomain, outputs the theta coordinates of $\Phi(P)$.
- SplittingIsomorphism: given theta null point 0_A on surface $A \cong E_1 \times E_2$, computes the isomorphism whose action on 0_A gives the theta null point associated with the product theta structure.

We defer the details of the algorithms to Chapter 8.

2.4.1. Computing a $(2,2)$-isogeny chain between products of elliptic curves

We now describe the computation of a chain of $(2,2)$-isogenies. Consider a $(2^e, 2^e)$-isogeny $\Phi : E_1 \times E_2 \longrightarrow E_3 \times E_4$ between products of elliptic curves. Knowing two generators P and Q of $\ker(\Phi)$, we explain how to compute Φ as a chain of $(2,2)$-isogenies:

$$E_1 \times E_2 \xrightarrow{\Phi_1} A_1 \xrightarrow{\Phi_2} A_2 \quad \cdots \quad A_{e-2} \xrightarrow{\Phi_{e-1}} A_{e-1} \xrightarrow{\Phi_e} E_3 \times E_4.$$

To compute each isogeny of the chain, it suffices to determine the theta null point of their codomain. Indeed, once this is known, the isogeny can be evaluated. A naïve method to compute of a chain of $(2,2)$-isogenies proceeds as follows:

(1) Compute the gluing isogeny $\Phi_1 : E_1 \times E_2 \longrightarrow A_1$ using the 8-torsion points $[2^{e-2}]P$ and $[2^{e-2}]Q$ lying above $\ker(\Phi_1) = \langle [2^{e-1}]P, [2^{e-1}]Q \rangle$ using GluingCodomain, and compute $\Phi_1(P), \Phi_2(P)$ using GluingEval.
(2) For all $i \in \{2, \cdots, e-2\}$, compute the generic isogeny $\Phi_i : A_{i-1} \longrightarrow A_i$ using the 8-torsion points $[2^{e-i-2}]\big(\Phi_{i-1} \circ \cdots \circ \Phi_1(P)\big)$ and $[2^{e-i-2}]\big(\Phi_{i-1} \circ \cdots \circ \Phi_1(Q)\big)$ lying above $\ker(\Phi_i)$ (see GenericCodomainWith8Torsion).
(3) Compute the generic isogeny $\Phi_{e-1} : A_{e-2} \longrightarrow A_{e-1}$ given the 4-torsion points $\Phi_{e-2} \circ \cdots \circ \Phi_1(P)$ and $\Phi_{e-2} \circ \cdots \circ \Phi_1(Q)$ lying above $\ker(\Phi_{e-1})$ (see GenericCodomainWith4Torsion).
(4) Compute the generic isogeny $\Phi_e : A_{e-1} \longrightarrow E_3 \times E_4$ given the 2-torsion kernel points $\Phi_{e-1} \circ \cdots \circ \Phi_1(P)$ and $\Phi_{e-1} \circ \cdots \circ \Phi_1(Q)$ (see GenericCodomain).
(5) Compute the change of coordinates from the system of theta coordinates on $E_3 \times E_4$ naturally induced by Φ_e to a system of product theta coordinates in order to express image points on $E_3 \times E_4$ in $(X : Z)$-Montgomery coordinates on each component E_3 and E_4 (see SplittingIsomorphism).

To avoid the square root computations in Steps (3) and (4), we can start with isotropic points $P, Q \in (E_1 \times E_2)[2^{e+2}]$ lying above $\ker(\Phi)$, i.e., such that $\ker(\Phi) = \langle [4]P, [4]Q \rangle$ and $e_{2^{e+2}}(P, Q) = 1$. Then, for all $i \in \{2, \cdots, e\}$, we compute $\Phi_i : A_{i-1} \longrightarrow A_i$ using the 8-torsion points $[2^{e-i}]\big(\Phi_{i-1} \circ \cdots \circ \Phi_1(P)\big)$ and $[2^{e-i}]\big(\Phi_{i-1} \circ \cdots \circ \Phi_1(Q)\big)$ lying above $\ker(\Phi_i)$ and GenericCodomainWith8Torsion. This optimization is only possible when 2^{e+2}-torsion points are defined over $\mathbb{F}_{p^2}$, which is not always the case.

Rather than following this naïve strategy, we use a balanced strategy. By storing intermediate points obtained during the doublings and pushing them through each isogeny, we can reduce the number of executions of the doubling algorithm ThetaDBL to a quasi-linear number $O(e \log(e))$. Note that we do not choose to use optimal strategies, which were first introduced in the context of SIDH/SIKE [JD11, Section 4.2.2], as they give only a small efficiency gain, but have a moderate memory cost due to the need to store the (precomputed) strategies.

2.4.1.1. Probability of failure. The computation of a $(2,2)$-isogeny chain may fail if we encounter a splitting before the final step. In Section 9.2 we argue this failure happens with probability $\tilde{O}(1/p)$ for the chains computed during key generation and signing. This failure does not happen during the verification of an honest signature, thus verification will simply reject if it encounters it.

2.4.1.2. Implementation details for chains of isogenies. In Section 2.4.1, we also give algorithmic details on how chains of isogenies are computed, building on the algorithms depicted for $(2,2)$-isogenies. In particular, we present the following algorithms:

- Isogeny22Chain: on input isotropic points $P, Q \in E_1 \times E_2$ of order 2^e, and an array `pts` containing points on $E_1 \times E_2$, outputs a $(2,2)$-isogeny chain $\Phi = \Phi_e \circ \cdots \circ \Phi_1$ such that $\ker(\Phi) = \langle P, Q \rangle$, and evaluated points $\{\Phi(P) : P \in \texttt{pts}\}$, computed using balanced strategies.
- Isogeny22ChainWithTorsion: on input isotropic points $P, Q \in E_1 \times E_2$ of order 2^{e+2}, and an array `pts` containing points on $E_1 \times E_2$, outputs a $(2,2)$-isogeny chain $\Phi = \Phi_e \circ \cdots \circ \Phi_1$ such that $\ker(\Phi) = \langle P, Q \rangle$, and evaluated points $\{\Phi(P) : P \in \texttt{pts}\}$, computed using balanced strategies.

More details of the algorithms can be found in Chapter 8.

CHAPTER 3

Basics of Quaternions

This chapter describes the world of quaternions, which we require for key-generation and signing. Specifically, verification only relies on the material in Chapter 2. This chapter describes quaternion algebras and their ideals (Section 3.1) and algorithms for moving between the world of quaternions and the world of isogenies (Section 3.2).

3.1. 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 ∞. Since these quaternion algebras are 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.

3.1.1. 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[1] is recommended. Future versions of this specification may determine the exact bounds on the largest representable integer and thus enable the use of fixed-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 integers;
- 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 XGCD: given (a, b), find integers (g, u, v) such that $ua + bv = g = \gcd(a, b) > 0$ and if both $a \neq 0$ and $b \neq 0$, then $1 \leq au \leq |ab|/g$ and $-|ab|/g < bv \leq 0$;
- Arithmetic modulo integers;
- Legendre symbol;
- DyadicValuation of an integer
- Square roots modulo primes.

With the exception of modular square roots (for which pseudocode is given in ModularSQRT), all these algorithms are implemented in GMP, which is the big integer library used by SQIsign reference implementation. The XGCD algorithm of GMP does however not enforce that $u \neq 0$ which is required in SQIsign.

[1] https://gmplib.org/

Algorithm 3.1 ModularSQRT(n, m)

Input: An odd prime m and an integer n such that n is a square modulo m.
Output: The modular square root x of n, i.e., an integer x such that $x^2 \equiv n \mod m$.

if $n \equiv 0 \mod m$ **then**
  return 0
if $m \equiv 3 \mod 4$ **then**
  return $n^{(m+1)/4} \bmod m$
if $m \equiv 5 \mod 8$ **then**
  if $n^{(m-1)/4} \not\equiv 1 \mod m$ **then**
    return $n^{(m+3)/8} \bmod m$
  else
    return $2n(4n)^{(m-5)/8} \bmod m$
// Now $m \equiv 1 \mod 8$, so use Tonelli-Shanks algorithm
$n \leftarrow 2$ and $e \leftarrow$ DyadicValuation$(p-1)$
$q \leftarrow (p-1)/2^e$
while w is a square modulo m **do**
  $w \leftarrow w + 1$
$z \leftarrow w^q \bmod m$ and $r \leftarrow e$ and $y \leftarrow n^q \bmod m$
$x \leftarrow n^{(q+1)/2} \bmod m$ and $f \leftarrow 2^{e-2}$
for i **from** 0 **up to** $e-1$ **do**
  $b \leftarrow y^f \bmod m$
  if $b = p - 1$ **then**
    $x \leftarrow xz \bmod m$
    $y \leftarrow xz^2 \bmod m$
  $z \leftarrow z^2 \bmod m$
  $f \leftarrow f/2$
return x

3.1.2. Floating-point numbers

SQIsign makes a limited use of floating-point numbers in the implementation of lattice reduction (see Section 3.1.4.1). The native floating-point types of most platforms are insufficient for SQIsign's needs, however any type with at least 24 bits of mantissa and at least 20 bits of exponent is largely sufficient for all security levels.

A standard way to build such a type is to pack a native floating-point type to hold the mantissa together with a native integer type to hold the exponent. The reference implementation uses the "double plus exponent" header library [PZ24] to this effect.

The basis output by a given lattice reduction algorithm implemented using floating-point numbers is highly sensitive to many factors, such as the exact order of operations performed (due to the non-associativity of floating-point numbers), the rounding mode in use, and possibly even the choice of compiler flags. Therefore, it may prove challenging for an alternative implementation of SQIsign to exactly reproduce the Known Answer Tests described in Chapter 6; however, this does not preclude the generation of valid signatures, which are verified by a correct implementation of Algorithm 4.9.

3.1.3. 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 $\mathbb{Z}$ or in $\mathbb{Z}/d\mathbb{Z}$ for some integer d.

3.1.3.1. 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.

3.1.3.2. Hermite normal form. The Hermite normal form (HNF) is a generalization of the reduced echelon form for matrices with integer coefficients. A matrix $\mathbf{H}$ 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 row is always strictly right of the leading coefficient of the row above 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.

Algorithm 3.2 $\mathsf{HNF}(\mathbf{M})$

Input: An integer matrix $\mathbf{M}$ with d rows and $c \geq d$ columns with rank d. Its columns are denoted by $\mathbf{M}_1^t$ to $\mathbf{M}_c^t$, and the coefficient in row r and column l is denoted by $\mathbf{M}_{r,l}$.
Output: The HNF of $\mathbf{M}$.

for i **from** d **down to** 1 **do**
  for j **from** $i-1$ **down to** 1 **do**
    if $\mathbf{M}_{i,i}$ and $\mathbf{M}_{i,j}$ are both 0 **then**
      $g, u, v \leftarrow 1, 1, 0$
    else
      $g, u, v \leftarrow \mathsf{XGCD}(\mathbf{M}_{i,i}, \mathbf{M}_{i,j})$ // After this step, $u\mathbf{M}_{i,i} + v\mathbf{M}_{i,j} = g$ and $u > 0$
    $\mathbf{M}_i^t \leftarrow u\mathbf{M}_i^t + v\mathbf{M}_j^t$ // After this step $\mathbf{M}_{i,i}$ equals g
  for j **from** $i-1$ **down to** 1 **do**
    $g \leftarrow \mathbf{M}_{i,j}/\mathbf{M}_{i,i}$ // g is an integer
    $\mathbf{M}_j^t \leftarrow \mathbf{M}_j^t - g\mathbf{M}_i^t$ // After this step $\mathbf{M}_{i,j} = 0$
  for j **from** $i+1$ **up to** c **do**
    $r \leftarrow \mathbf{M}_{i,j} \bmod \mathbf{M}_{i,i}$ // After this r is in $[0, \mathbf{M}_{i,i} - 1]$
    $g \leftarrow (\mathbf{M}_{i,j} - r)/\mathbf{M}_{i,i}$ // g is an integer
    $\mathbf{M}_j^t \leftarrow \mathbf{M}_j^t - g\mathbf{M}_i^t$ // After this step $\mathbf{M}_{i,j} = r \in [0, \mathbf{M}_{i,i} - 1]$
return $\mathbf{M}$

A matrix $\mathbf{A}$ is said to have $\mathbf{H}$ for HNF if $\mathbf{H}$ is in HNF and there exists a *unimodular* matrix $\mathbf{U}$ such that $\mathbf{AU} = \mathbf{H}$, i.e., $\mathbf{U}$ has integer coefficients and determinant ± 1. Then, $\mathbf{A}$ and $\mathbf{H}$ have the same column space, and $\mathbf{H}$ is unique, thus giving a canonical representation for $\mathbf{A}$'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, 4×8 and 4×16 integer matrices, all of which have rank 4. An example of an algorithm computing the HNF for full-rank matrices is given in Algorithm 3.2. Any efficient algorithm producing the same output can be used equivalently.

3.1.4. Lattices

A *symmetric bilinear form* on a $\mathbb{Q}$-vector space V is a mapping that to any $\mathbf{a}, \mathbf{b} \in V$ associates a value $\langle \mathbf{a}, \mathbf{b} \rangle \in \mathbb{Q}$ with the properties

- $\langle \mathbf{a}, \mathbf{b} \rangle = \langle \mathbf{b}, \mathbf{a} \rangle$,
- $\langle \mathbf{a} + \mathbf{b}, \mathbf{c} \rangle = \langle \mathbf{a}, \mathbf{c} \rangle + \langle \mathbf{b}, \mathbf{c} \rangle$,
- $\langle \lambda\mathbf{a}, \mathbf{b} \rangle = \lambda \langle \mathbf{a}, \mathbf{b} \rangle$,

for any $\mathbf{a}, \mathbf{b} \in V$ and any $\lambda \in \mathbb{Q}$. A finite-dimensional vector space equipped with a symmetric bilinear form is called a *quadratic space*. Two elements $\mathbf{a}, \mathbf{b}$ are said *orthogonal* when $\langle \mathbf{a}, \mathbf{b} \rangle = 0$. Every quadratic space admits an *orthogonal basis*, i.e., a basis such that any two elements are orthogonal.

To any symmetric bilinear form is associated a *quadratic form* defined by $Q(\mathbf{a}) = \langle \mathbf{a}, \mathbf{a} \rangle$. If $Q(\mathbf{a}) > 0$ for all $\mathbf{a} \neq 0$, the quadratic form is called *positive definite* and so are the bilinear form and the quadratic space. In this case we call $Q(\mathbf{a})$ the *length* of the vector $\mathbf{a}$. An example of positive definite symmetric bilinear form is the inner product $\mathbf{a} \cdot \mathbf{b}$ of $\mathbb{Q}^n$, its associated quadratic form being the squared Euclidean norm $\|\mathbf{a}\|^2$.

A *lattice* of V is the set of all integer linear combinations of a basis of V. Two bases of V generate the same lattice if and only if the change-of-basis matrix is unimodular. Given a basis $\mathbf{b}_0, \ldots, \mathbf{b}_{d-1}$ of a quadratic space, its *Gram matrix* is the symmetric matrix whose (i, j)-th entry is $\langle \mathbf{b}_i, \mathbf{b}_j \rangle$. This matrix uniquely determines the bilinear form on the lattice spanned by $\mathbf{b}_0, \ldots, \mathbf{b}_{d-1}$; in particular, it is diagonal if and only if the basis is orthogonal. Lattices of quadratic spaces do not always admit an orthogonal basis. The goal of *lattice reduction* is to compute a basis of a lattice that is "as orthogonal as possible" in a precise sense.

3.1.4.1. Lattice reduction.

Definition 3.1.1 (Reduced basis)**.** Let $\mathbf{b}_0, \ldots, \mathbf{b}_{d-1}$ be a basis of a postive definite quadratic space. Define the *Gram-Schmidt vectors* $\overline{\mathbf{b}}_i$ as

$$\overline{\mathbf{b}}_i = \mathbf{b}_i - \sum_{j=0}^{i-1} \mu_{i,j} \overline{\mathbf{b}}_j, \qquad \text{where} \qquad \mu_{i,j} = \frac{r_{i,j}}{r_{j,j}} \qquad \text{and} \qquad r_{i,j} = \left\langle \mathbf{b}_i, \overline{\mathbf{b}}_j \right\rangle. \tag{8}$$

The basis is said to be (η, δ)-reduced for parameters $\frac{1}{2} < \eta < 1$ and $\frac{1}{4} < \delta < 1$ if:

- $|\mu_{i,j}| < \eta$ for $0 \leq j < i < d$, and
- $\left\langle \overline{\mathbf{b}}_i, \overline{\mathbf{b}}_i \right\rangle \geq (\delta - \mu_{i,i-1}^2) \left\langle \overline{\mathbf{b}}_{i-1}, \overline{\mathbf{b}}_{i-1} \right\rangle$ for $1 \leq i < d$.

A symmetric matrix is said to be (η, δ)-reduced if it is the Gram matrix of an (η, δ)-reduced basis.

The L2 algorithm of [NS09] takes as input a basis of a lattice and its associated Gram matrix and outputs an (η, δ)-reduced basis for the same lattice, along with its associated Gram matrix. This algorithm only applies (multiple precision) integer operations on the lattice basis and the Gram matrix. Internally, however, it uses floating-point operations to keep track of the values $r_{i,j}$ and $\mu_{i,j}$ for $i \geq j$, called the *GSO family* in [NS09] (see also [NV10, Chapter 5]). We reproduce pseudocode for L2 in Algorithm 3.3. Numbers, vectors and matrices that are stored as floating-point values are indicated by the type annotation :: FLOAT. Similarly, all casts from integer to floating point are explicitly indicated by the operator FLOAT().

Algorithm 3.3 $\mathsf{L2}_{\eta,\delta}((\mathbf{b}_0, \ldots, \mathbf{b}_{d-1}), \mathbf{G})$

Input: A basis $(\mathbf{b}_0, \ldots, \mathbf{b}_{d-1})$ of a d-dimensional quadratic space, the associated $d \times d$ Gram matrix $\mathbf{G}$.
Parameters: $\frac{1}{2} < \eta < 1$, $\frac{1}{4} < \delta < 1$.
Output: A (η, δ)-reduced bases of the same lattice, its associated Gram matrix.
$\bar{\delta} \leftarrow \text{FLOAT}\left(\frac{\delta+1}{2}\right)$, $\bar{\eta} \leftarrow \text{FLOAT}\left(\frac{\eta+0.5}{2}\right)$
$r_{0,0} \leftarrow \text{FLOAT}(\mathbf{G}_{0,0})$, $\mu_{0,0} \leftarrow \text{FLOAT}(1)$, $T \leftarrow [\text{FLOAT}(0), \text{FLOAT}(0), \text{FLOAT}(0), \text{FLOAT}(0)]$
$k = 1$
while $k < d$ **do**
  $(\mathbf{b}_0, \ldots, \mathbf{b}_{d-1}), \mathbf{G}, r, \mu \leftarrow \mathsf{SizeReduce}((\mathbf{b}_0, \ldots, \mathbf{b}_{d-1}), \mathbf{G}, k, r, \mu, \bar{\eta})$
  $T_0 \leftarrow \text{FLOAT}(\mathbf{G}_{k,k})$
  for i **from** 1 **up to** $k - 1$ **do**
    $T_i \leftarrow T_{i-1} - \mu_{k,(i-1)} r_{k,(i-1)}$
  $s \leftarrow \min\{0 \leq i \leq k \text{ such that } T_j < \bar{\delta} r_{j,j} \text{ for all } i \leq j < k\}$
  if $k \neq s$ **then**
    $(\mathbf{b}_0, \ldots, \mathbf{b}_{d-1}), \mathbf{G}, r, \mu \leftarrow \mathsf{InsertBefore}((\mathbf{b}_0, \ldots, \mathbf{b}_{d-1}), \mathbf{G}, k, s, r, \mu)$
    $k \leftarrow s$
  $k \leftarrow k + 1$
return $(\mathbf{b}_0, \ldots, \mathbf{b}_{d-1}), \mathbf{G}$

SQISIGN only manipulates quadratic spaces and lattices of dimension 4. The elements of the space are integer vectors of length 4 themselves, typically represented as column vectors. Thus a lattice basis is conveniently represented by a 4×4 integer matrix, and the operations in SizeReduce and InsertBefore are easily understood as multiplying it by elementary unimodular matrices.

Algorithm 3.4 ExtendGSOFamily($\mathbf{G}, k, r :: \text{FLOAT}, \mu :: \text{FLOAT}$)

Input: $\mathbf{G}$ a $d \times d$ Gram matrix, index $1 \leq k < d$, the GSO family r, μ up to row $k-1$.
Output: The GSO family r, μ up to row k.

for j **from** 0 **up to** k **do**
  $r_{k,j} \leftarrow \text{FLOAT}(\mathbf{G}_{k,j})$
  for l **from** 0 **up to** $j-1$ **do**
    $r_{k,j} \leftarrow r_{k,j} - r_{k,l}\mu_{j,l}$
  if $j < k$ **then**
    $\mu_{k,j} \leftarrow \frac{r_{k,j}}{r_{j,j}}$
return r, μ

Algorithm 3.5 SizeReduce($(\mathbf{b}_0, \ldots, \mathbf{b}_{d-1}), \mathbf{G}, k, r :: \text{FLOAT}, \mu :: \text{FLOAT}, \bar{\eta} :: \text{FLOAT}$)

Input: A basis $(\mathbf{b}_0, \ldots, \mathbf{b}_{d-1})$ of a quadratic space, its $d \times d$ Gram matrix $\mathbf{G}$, index $1 \leq k < d$, the GSO family r, μ up to row $k-1$, parameter $\frac{1}{2} < \bar{\eta} < 1$.
Output: $(\mathbf{b}_0, \ldots, \mathbf{b}_{d-1})$ size-reduced basis of the same lattice, its Gram matrix $\mathbf{G}$, the GSO family r, μ up to row k.

done $\leftarrow$ `false`
while not done **do**
  $r, \mu \leftarrow$ ExtendGSOFamily$(\mathbf{G}, k, r, \mu)$
  done $\leftarrow$ `true`
  for i **from** $k-1$ **down to** 0 **do**
    if $|\mu_{k,i}| > \bar{\eta}$ **then**
      done $\leftarrow$ `false`
      $X \leftarrow \lfloor \mu_{k,i} \rceil$ // Round to the closest integer
      $\mathbf{b}_k \leftarrow \mathbf{b}_k - X\mathbf{b}_i$ // Update basis
      for j **from** 0 **up to** $d-1$ **do**
        $\mathbf{G}_{k,j} \leftarrow \mathbf{G}_{k,j} - X\mathbf{G}_{i,j}$ // Update Gram matrix
      for j **from** 0 **up to** $d-1$ **do**
        $\mathbf{G}_{j,k} \leftarrow \mathbf{G}_{j,k} - X\mathbf{G}_{j,i}$ // Update Gram matrix
      for j **from** 0 **up to** $i-1$ **do**
        $\mu_{k,j} \leftarrow \mu_{k,j} - \text{FLOAT}(X)\mu_{i,j}$ // Update μ
return $(\mathbf{b}_0, \ldots, \mathbf{b}_{d-1}), \mathbf{G}, r, \mu$

Algorithm 3.6 InsertBefore($(\mathbf{b}_0, \ldots, \mathbf{b}_{d-1}), \mathbf{G}, k, s, r :: \text{FLOAT}, \mu :: \text{FLOAT}$)

Input: A basis $(\mathbf{b}_0, \ldots, \mathbf{b}_s, \ldots, \mathbf{b}_k, \ldots, \mathbf{b}_{d-1})$ of a quadratic space, its $d \times d$ Gram matrix $\mathbf{G}$, indices $0 \leq s < k < d$, the GSO family r, μ up to row k.
Output: The basis $(\mathbf{b}_0, \ldots, \mathbf{b}_k, \mathbf{b}_s, \ldots \mathbf{b}_{d-1})$ where $\mathbf{b}_k$ has been inserted before $\mathbf{b}_s$, the associated Gram matrix, the GSO family r, μ up to row s.

for j **from** k **down to** $s+1$ **do**
  swap $\mathbf{b}_j$ and $\mathbf{b}_{j-1}$
  for i **from** 0 **up to** $d-1$ **do**
    swap $\mathbf{G}_{i,j}$ and $\mathbf{G}_{i,j-1}$
  for i **from** 0 **up to** $d-1$ **do**
    swap $\mathbf{G}_{j,i}$ and $\mathbf{G}_{j-1,i}$
$r_{s,s} \leftarrow \text{FLOAT}(\mathbf{G}_{s,s})$
for i **from** 0 **up to** $s-1$ **do**
  $\mu_{s,i} \leftarrow \mu_{k,i}$
  $r_{s,i} \leftarrow r_{k,i}$
  $r_{s,s} \leftarrow r_{s,s} - \mu_{s,i} r_{s,i}$
return $(\mathbf{b}_0, \ldots, \mathbf{b}_{d-1}), \mathbf{G}, r, \mu$

	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 $+\infty$.

The parameters η and δ have relatively little impact on the performance of SQISIGN, as long as they are taken close to $1/2$ and 1 respectively. Following standard practice [FPLLL], they can be taken as $\eta = 0.51, \delta = 0.99$.

Regarding the floating-point operations, [NS09, Theorem 2] shows that 24 bits of mantissa are largely sufficient for all three security levels of SQISIGN. On the other hand, exponents at least as large as the volume of the lattice are required, which is why we recommend to use at least 20 bits of exponent for implementing the floating-point operations.

3.1.4.2. Lattice enumeration and sampling. Given a lattice Λ generated by a basis $(\mathbf{b}_0, \dots, \mathbf{b}_{d-1})$ of a positive definite quadratic space, two related tasks are enumerating all vectors of Λ of length less than a given bound and uniformly sampling from the same set. A common solution to both consists in determining a parallelogram containing all the vectors of interest and enumerating/sampling from the parallelogram, rejecting the vectors whose length surpasses the bound.

SQISIGN only needs to implement the sampling variant of the algorithm. For this, we adapt in LatticeSampling an enumeration algorithm of Dieter [Die75]. For reference, we prove in Appendix A that our algorithm is correct and argue that, under mild heuristics, it terminates in a small number of iterations of the main loop. Better algorithms exist for enumeration [FP85], however they are not easily adapted to sampling uniformly.

Algorithm 3.7 LatticeSampling$((\mathbf{b}_0, \dots, \mathbf{b}_{d-1}), \mathbf{G}, B)$

Input: A basis $(\mathbf{b}_0, \dots, \mathbf{b}_{d-1})$ of a quadratic space, its $d \times d$ Gram matrix $\mathbf{G}$, an integer $B > 0$.
Output: A uniformly random vector of length $\leq B$ in the lattice generated by $(\mathbf{b}_0, \dots \mathbf{b}_{d-1})$.

Initialize $\mathbf{Id}$ to the $d \times d$ identity matrix
$\mathbf{U}, \mathbf{H} \leftarrow \mathsf{L2}_{\eta,\delta}(\mathbf{Id}, \mathbf{G}^{-1})$ // Parse columns of $\mathbf{U}$ and $\mathbf{Id}$ as basis vectors
repeat
  for i **from** 0 **up to** $d-1$ **do**
    $B_i \leftarrow \lfloor \sqrt{B\mathbf{H}_{i,i}} \rfloor$
    Sample x_i uniformly from $[-B_i, B_i]$
  $(y_0 \cdots y_{d-1}) \leftarrow (x_0 \cdots x_{d-1})\mathbf{U}^{-1}$
until $(y_0 \cdots y_{d-1})\mathbf{G}(y_0 \cdots y_{d-1})^t \leq B$
return $y_0\mathbf{b}_0 + \cdots + y_{d-1}\mathbf{b}_{d-1}$

3.1.5. Quaternions and ideal 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 $(\alpha_1, \dots, \alpha_4)$ is basis of $B_{p,\infty}$ as a $\mathbb{Q}$-vector space. We now describe how these objects are represented and manipulated.

3.1.5.1. Basic quaternion arithmetic. We already mentioned that 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 Section 3.1.5.1.

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

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

The map

$$\langle \alpha, \beta \rangle = \mathrm{tr}(\alpha\bar{\beta}) \tag{9}$$

is a symmetric bilinear form, turning $B_{p,\infty}$ into a quadratic space. Additionally

$$\langle \alpha, \alpha \rangle = \mathrm{tr}(\alpha\bar{\alpha}) = 2\,\mathrm{nrd}(\alpha),$$

implying that the quadratic space is positive definite.

3.1.5.2. Quaternion 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.[2]

The dual of a lattice Λ is defined as

$$\Lambda^\star = \{f \in B_{p,\infty}^\star \mid \forall x \in \Lambda, f(x) \in \mathbb{Z}\},$$

where $B_{p,\infty}^\star$ denotes the space of linear functions $B_{p,\infty} \to \mathbb{Q}$. Writing $(1^\star, i^\star, j^\star, k^\star)$ for the basis of the dual lattice $(\mathbb{Z} + i\,\mathbb{Z} + j\,\mathbb{Z} + k\,\mathbb{Z})^\star$, if Λ is generated by a basis $(1\ i\ j\ k) \cdot L$, then $\Lambda^\star$ is generated by $L^{-1} \cdot (1^\star\ i^\star\ j^\star\ k^\star)^t$.

In what follows we are going to abuse notation and identify the columns (resp. rows) of a matrix defining a lattice basis with the lattice (resp. dual lattice) itself. Basic operations are computed as follows.

Equality: Check that L_1 and L_2 have the same HNF [Coh93, § 2.4.3].

Sum: 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 a basis $\alpha_1, \ldots, \alpha_4$ of L_2, and then computing the sum

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

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.

Inclusion: Checking whether a lattice L_1 is included in a lattice L_2, can be done either by checking containment of all 4 basis vectors of L_1 in L_2, or by testing equality of $L_1 + L_2$ and L_2.

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 can be computed using the determinant algorithm for dimension 4 from [Ebe08].

Basis reduction: We say a lattice is *reduced* when its basis is reduced according to the definition of Section 3.1.4.1 instantiated with the bilinear form of Eq. (9). We use the L2 algorithm to compute a reduced basis. The Gram matrix of the basis $(1, i, j, ij)$ is the diagonal matrix $\mathbf{G}$ of entries $(2, 2, 2p, 2p)$, and the Gram matrix of any other basis is computed as $\mathbf{G}' = \mathbf{M}^t\mathbf{G}\mathbf{M}$ if $\mathbf{M}$ is the change-of-basis matrix.

3.1.6. 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.

The quaternion algebra used in SQIsign contains a maximal order with basis $(1, i, \frac{i+j}{2}, \frac{1+ij}{2})$ which will be denoted by $\mathcal{O}_0$ in the remainder of this section. $\mathcal{O}_0$ contains a (non-maximal) suborder of basis $(1, i, j, ij)$.

Let $\mathcal{O}$ be an order. A left (right) *integral ideal* of $\mathcal{O}$, henceforth only called a left (right) ideal of $\mathcal{O}$, is a sublattice of $\mathcal{O}$, closed under multiplication by $\mathcal{O}$ on the left (right). 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 $\mathcal{O}_R(I)$. In this case, I is clearly 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

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

an ideal I, denoted by $\mathrm{nrd}(I)$, 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 product of I and J as lattices. 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 $\beta\mathcal{O}_R(I) = \mathcal{O}_R(J)\beta$.

3.1.6.1. Basic operations on ideals. We represent ideals by their lattices and compute equality, membership, sum, 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 sum to compute $\mathcal{O}\langle\alpha, N\rangle$. Inversely, a quaternion $\alpha \in I$ 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. This method is used in the reference implementation and detailed in Algorithm 3.8.

Ideal inverse: Since all ideals in SQIsign are connecting maximal orders, they have inverses. The inverse of such an ideal I is $I^{-1} = \frac{1}{\mathrm{nrd}(I)}\bar{I}$ where $\bar{I}$ is the lattice of conjugates of elements in I. I^{-1} is not an ideal, but a rank-4 lattice, and for lattice multiplication, $II^{-1} = \mathcal{O}_L(I)$ and $I^{-1}I = \mathcal{O}_R(I)$.

The left and right order of an ideal: Since all ideals in SQIsign are ideals connecting maximal orders, the left and right orders of an ideal I can be computed using its inverse, since $\mathcal{O}_L(I) = II^{-1}$ and $\mathcal{O}_R(I) = I^{-1}I$.

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

$$I = N\mathcal{O}_L\mathcal{O}_R,$$

where N is the square root of the index of $\mathcal{O}_L \cap \mathcal{O}_R$ in $\mathcal{O}_L$, or, alternatively, a common denominator of the lattice $\mathcal{O}_L\mathcal{O}_R$.

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 3.2), where the corresponding definitions are more natural.

3.1.6.2. Finding random ideals. SQIsign requires to compute two different kinds of random ideals:

(1) An ideal J which is equivalent to a given ideal I as left $\mathcal{O}_L(I)$-ideal and has norm $\mathrm{nrd}(I)$ prime and rather small. J is sampled randomly among such ideals of small norm. This is done by Algorithm 3.9. The distribution of the output J among all such ideals does not matter for the security of the scheme as this algorithm will only be used to compress the secret key and commitment ideal representations.
(2) A left ideal J' of $\mathcal{O}_0$ of norm $\mathrm{nrd}(J')$ equal to a given integer n of sufficient size. J' must be sampled uniformly at random among all such ideals. This is described in Algorithm 3.10.

We now describe these algorithms in more detail.

RandomEquivalentPrimeIdeal**.** To be more precise, given an integral ideal I, RandomEquivalentPrimeIdeal finds an equivalent ideal J (i.e., $I = J\alpha$, where $\alpha \in \mathcal{B}^\star_{p,\infty}$) with different (i.e., prime and bounded) norm. To do so, it employs the surjection

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

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

Algorithm 3.8 IdealGenerator(I)

Input: I, an ideal.
Output: γ primitive in I such that its norm N_I and γ generate I as ideal of its left or right order.

$n \leftarrow 0$
while `true` **do**
  $n \leftarrow n + 1$
  for a **from** $-n$ **up to** n **do**
    for b **from** $-n + |a|$ **up to** $n - |a|$ **do**
      for c **from** $-n + |a| + |b|$ **up to** $n - |a| - |b|$ **do**
        $d \leftarrow n - |a| - |b| - |c|$
        if $\gcd(a, b, c, d) = 1$ **then**
          // The quaternion γ has a denominator that divides the ideal's denominator
          $\gamma \leftarrow$ The quaternion of coordinates a, b, c, d in the basis in HNF of I
          $q = \mathrm{nrd}(\gamma)/N_I$
          if $\gcd(q, N_I) = 1$ **then**
            return γ

The algorithm samples constants c_i and constructs $\beta = \sum_{i=1}^{4} c_i\alpha_i$, where $(\alpha_1, \ldots, \alpha_4)$ is a reduced basis of I, before outputting $J = \chi_I(\beta)$ if the norm of J is prime. Heuristically, we can expect the norm of the output ideal to be $\approx \sqrt{p}$ which should be much smaller than the norm of the input in SQIsign.

Algorithm 3.9 RandomEquivalentPrimeIdeal(I)

Input: I, a left $\mathcal{O}$-ideal.
Output: $J \sim I$ of small prime norm, or raise an exception if unsuccessful.

Initialize `counter` $\leftarrow 0$
Compute a LLL-reduced basis $\alpha_1, \alpha_2, \alpha_3, \alpha_4$ of I
while `counter` $< (2 \cdot$ `QUAT_equiv_bound_coeff` $+ 1)^4$ **do**
  `counter` $\leftarrow$ `counter` $+ 1$
  Sample c_1, c_2, c_3, c_4 uniformly at random from $[-b, \ldots, b]$, for bound $b =$ `QUAT_equiv_bound_coeff`
  $\beta \leftarrow \sum_{i=1}^{4} c_i\alpha_i$
  $J \leftarrow \chi_I(\beta)$
  if $\mathrm{nrd}(J)$ is prime **then**
    return J
raise Exception: ("RandomEquivalentPrimeIdeal failed")

RandomIdealGivenNorm**.** RandomIdealGivenNorm first uses GeneralizedRepresentInteger described below to find an element γ of $\mathcal{O}_0$ of norm a multiple of the given integer N (N cannot be a multiple of p). If GeneralizedRepresentInteger fails, it propagates the error. It then multiplies this quaternion with a uniformly random quaternion β in $\mathcal{O}_0/N\mathcal{O}_0$ of norm coprime to N. The result $\gamma\beta$ still has norm divisible by N. The left $\mathcal{O}_0$-ideal generated by N and $\gamma\beta$ is then returned by the algorithm.

In order to use GeneralizedRepresentInteger, it is needed to compute a multiple of N coprime to p, with few prime factors, and of size at least $p \cdot$ `QUAT_repres_bound_input`. Therefore RandomIdealGivenNorm needs first to find a prime distinct from p but of similar size. In SQIsign, the smallest prime of the same bit size as p is precomputed and used. This prime m is multiplied to N to create a suitable input for GeneralizedRepresentInteger.

In case N is known to be prime, RandomIdealGivenNorm should be called with the parameter `prime` set to true. In this case, the sampling doesn't use GeneralizedRepresentInteger but instead samples a random quaternion of trace zero and norm n with $-n$ square modulo N.

3.1.7. Solving norm equations

An important part of both the key generation and signing procedure in SQIsign, is the ability to solve norm equations in special quaternion orders, sometimes with some additional constraints. Throughout this section, we consider special extremal maximal orders. These are maximal orders that contain j and a distinguished quadratic subring R of small discriminant, such that R and jR are orthogonal.

Algorithm 3.10 RandomIdealGivenNorm(N, `prime`)

Input: A positive integer N not multiple of p which is the norm of some left $\mathcal{O}_0$ ideal.
Input: A boolean `prime` indicating whether N is prime.
Input: A fixed parameter $m =$ `QUAT_prime_cofactor` which is prime, approximately as large as p and larger than N but distinct from both.
Output: A random left ideal J' of $\mathcal{O}_0$ of norm N, or raise an exception.

`found` $\leftarrow$ `false`
if `prime` **then**
  while not `found` **do**
    $g_1, g_2, g_3 \leftarrow$ independent uniformly random integers in $[0, N-1]$
    $\gamma \leftarrow g_1 i + g_2 j + g_3 ij$
    `found` $\leftarrow (1 = \text{Legendre}(-\text{nrd}(\gamma), N))$
    if `found` **then**
      $\gamma \leftarrow \gamma + \text{ModularSQRT}(-\text{nrd}(\gamma), N)$
else
  $\gamma \leftarrow \text{GeneralizedRepresentInteger}(mN, i, \mathcal{O}_0, \texttt{false})$ // This might raise an exception
while not found **do**
  x, y, z, w uniformly randomly selected integers in $[1, N]$
  $\beta \leftarrow x + yi + zj + wij$
  found $\leftarrow (\gcd(\text{nrd}(\beta), N) = 1)$
$J' \leftarrow$ the left $\mathcal{O}_0$-ideal generated by $\gamma\beta$ and N
return J'

3.1.7.1. Cornacchia's algorithm. Cornacchia's algorithm [Cor08] allows us to efficiently find integer solutions for equations of the form $x^2 + qy^2 = m$ with q,m positive integers, provided we know the factorization of m. For prime m, a pseudocode following [MN90] is given in Algorithm 3.11.

Algorithm 3.11 Cornacchia(q, m)

Input: $q, m \in \mathbb{Z}$ with m prime and $0 \leq q \leq m$.
Output: $x, y \in \mathbb{Z}$ such that $x^2 + qy^2 = m$ if such x, y exist, non-existence indicator $\perp$ otherwise.

if $-q$ is not a square modulo m **then**
  return $\perp$
if $m = 2$ **then**
  if $q = 1$ **then**
    return $1, 1$
  else
    return $\perp$
// Now m is an odd prime and $-q$ square modulo m
$r \leftarrow \text{ModularSQRT}(-q \bmod m, m)$
$s \leftarrow q$
$r, s \leftarrow s, (r \bmod s)$
$r, s \leftarrow s, (r \bmod s)$
while $s^2 < m \leq r^2$ **do**
  $r, s \leftarrow s, (r \bmod s)$
$x, y \leftarrow s, (m - s^2)/q$
if $x^2 + qy^2 = m$ **then**
  return r, s
else
  return $\perp$

3.1.7.2. Representing integers by a special extremal order. Cornacchia's algorithm efficiently solves norm equations of the form $x^2 + ny^2 = m$. This makes it straightforward to solve norm equations in special extremal orders, i.e., orders containing a suborder of the form $\mathbb{Z}[\omega] + j\,\mathbb{Z}[\omega]$, where $j^2 = -p$ and $\mathbb{Z}[\omega]$ is a quadratic subring

of $B_{p,\infty}$ of small discriminant. The generator ω is chosen as $\sqrt{-q}$ for small positive integer q. The norm of an element of coefficients (x, y, z, t) in basis $1, \omega, j, \omega j$ is then

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

The general idea for solving equations of the form $F(x, y, z, t) = M$ for some target norm m is to sample z, t, and then try to solve for x, y such that $x^2 + qy^2 = M - p(z^2 + qt^2)$ using Cornacchia.

For example, $\mathcal{O}_0$ is a special extremal maximal order with $q = 1$ and $\omega = i$. Solving norm equations in its suborder $\mathbb{Z}[i] + j\,\mathbb{Z}[i]$ can therefore be done as described above. An extension of that algorithm was introduced in [DLLW23, §6], which samples solutions in the whole order $\mathcal{O}_0$. We further generalize this algorithm to other special extremal maximal orders of $B_{p,\infty}$ in GeneralizedRepresentInteger.

When called on $\mathcal{O}_0$ with the boolean `isogenyCond` set to `true`, GeneralizedRepresentInteger enforces an additional modular condition on the solution, as required by FixedDegreeIsogeny for constructing two-dimensional isogenies.

The algorithm GeneralizedRepresentInteger only works on odd input norms M and is likely to fail if $M < pq$`QUAT_repres_bound_input`, and less efficient if q is too large. See Chapter 9 for more details on heuristics and failures.

Algorithm 3.12 GeneralizedRepresentInteger$(M, \omega, \mathcal{O}$, `isogenyCond`$)$

Input: $M \in \mathbb{Z}$ odd such that $M > p$.
Input: $\omega \in B_{p,\infty}$ such that such that $q = -\omega^2$ is a positive integer and $q = 1 \bmod 4$.
Input: $\mathcal{O}$ a special extremal maximal order containing the suborder $\mathbb{Z}[\omega] + j\,\mathbb{Z}[\omega] \subset \mathcal{O}$, with j from the standard basis of $B_{p,\infty}$.
Input: `isogenyCond` boolean flag only relevant for $-\omega^2 = 1$, enforces a modular condition needed by FixedDegreeIsogeny.
Output: $\gamma \in \mathcal{O}$ with $\mathrm{nrd}(\gamma)$ equal to M, or raises an exception.

Initialize $q \leftarrow -\omega^2$, `counter` $\leftarrow 0$, `bound` $\leftarrow \left\lceil \frac{4M}{p\sqrt{q}} \right\rceil$, and `found` $\leftarrow$ `false`
while (**not** `found`) **and** (`counter` $<$ `bound`) **do**
  `counter` $\leftarrow$ `counter` $+ 1$
  Sample z uniformly from $[1, \ldots, m]$ for $m = \left\lfloor \sqrt{\frac{4M}{p} - q} \right\rfloor$
  Sample t uniformly from $[-m', \ldots, m']$ for $m' = \left\lfloor \sqrt{\frac{4M - pz^2}{qp}} \right\rfloor$
  Set $M' \leftarrow 4M - p(z^2 + qt^2)$
  if M' is a prime **then**
    `res` $\leftarrow$ Cornacchia(q, M')
    `found` $\leftarrow$ (`res` $\neq \perp$)
    if `found` **then**
      $x, y \leftarrow$ `res`
    if `found` **and** `isogenyCond` **and** $q = 1$ **then**
      if $x \not\equiv t \mod 2$ **then**
        Swap x and y
      `found` $\leftarrow (x - t \equiv 2 \mod 4)$ **and** $(y - z \equiv 2 \mod 4)$
    if `found` **then**
      $\gamma \leftarrow (x + \omega y + jz + \omega jt)$
      Set d to be the biggest scalar such that $\gamma/d \in \mathcal{O}$
      `found` $\leftarrow (d = 2)$
    if `found` **then**
      return γ/d
raise Exception: ("GeneralizedRepresentInteger failed")

3.2. Converting between ideals and isogenies

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

3.2.1. 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

$$E[I] := \{P \in E \mid \alpha(P) = 0_E, \forall \alpha \in I\},$$

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

$$I_\varphi = \{\alpha \in \mathcal{O} \mid \alpha(P) = 0_E, \forall P \in \ker \varphi\}$$

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

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

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 $\mathcal{O} \cong \mathrm{End}(E)$.

3.2.1.1. Precomputed data. In the following algorithms, we will use p of the form $c \cdot 2^f - 1$, where c is a small integer. Let (P_0, Q_0) be a basis of $E_0[2^f]$, where the actions of i, $\frac{i+j}{2}$, $\frac{1+k}{2}$ are precomputed.

For an index t, let $(\mathcal{O}_t, J_t, E_t, (P_t, Q_t), (\mathbf{M}_{t,1}, \ldots, \mathbf{M}_{t,4}))$ be a tuple of the following form:

- $\mathcal{O}_t = b_{t,1}\,\mathbb{Z} \oplus b_{t,2}\,\mathbb{Z} \oplus b_{t,3}\,\mathbb{Z} \oplus b_{t,4}\,\mathbb{Z}$ is a maximal order in $B_{p,\infty}$ containing a subring R isomorphic to $\mathbb{Z}[\sqrt{-q_t}]$ for a suitable small integer q_t, such that $R + jR$ is a rank-4 submodule of $\mathcal{O}_t$,
- J_t is an $(\mathcal{O}_0, \mathcal{O}_t)$-ideal of odd norm,
- E_t is a supersingular elliptic curve over $\mathbb{F}_p$ such that $\mathrm{End}(E_t) \cong \mathcal{O}_t$,
- (P_t, Q_t) is the basis of $E_t[2^f]$ such that $\varphi_{J_t}(P_0) = P_t, \varphi_{J_t}(Q_0) = Q_t$,
- $\mathbf{M}_{t,u}$ is the matrix in $\mathbb{M}_2(\mathbb{Z}/2^f\,\mathbb{Z})$ representing the action of $b_{t,u}$ on $E_t[2^f]$ with respect to the basis (P_t, Q_t) for $u \in \{1, \ldots 4\}$.

For $\alpha = x_1 b_{t,1} + x_2 b_{t,2} + x_3 b_{t,3} + x_4 b_{t,4} \in \mathcal{O}_t$, we can compute the action of α on $E_t[2^f]$ by computing the matrix $\mathbf{M}_\alpha = x_1\mathbf{M}_{t,1} + x_2\mathbf{M}_{t,2} + x_3\mathbf{M}_{t,3} + x_4\mathbf{M}_{t,4}$.

We define $q_0 = 1$ and $J_0 = \mathcal{O}_0$ and $\mathbf{M}_{0,1}, \ldots, \mathbf{M}_{0,4}$ as the same as the above for the integral basis $(1, i, \frac{i+j}{2}, \frac{1+k}{2})$ of $\mathcal{O}_0$. We precompute distinct tuples $(\mathcal{O}_t, J_t, E_t, (P_t, Q_t), (\mathbf{M}_{t,1}, \ldots, \mathbf{M}_{t,4}))$ for $t = 0, 1, \ldots, n_{\text{orders}}$, and use them in the following algorithms.

The precomputed tuples may be generated as follows: First, generate the tuple associated to $\mathcal{O}_0$. Then, for all $t = 1, \ldots, n_{\text{orders}}$, look for the smallest prime $q_t > q_{t-1}$ with $q_t \equiv 1 \pmod 4$ such that $j^2 = -p,\ i^2 = -q_t$ defines a quaternion algebra of discriminant p. The order $\mathcal{O}_t$ is obtained as

$$\mathcal{O}_t = \mathbb{Z} + i\,\mathbb{Z} + \frac{1+j}{2}\,\mathbb{Z} + \frac{(r+j)i}{2q_t}\,\mathbb{Z}$$

where $r \in \mathbb{Z}$ is the smallest positive integer such that $r^2 \equiv -p \pmod{q_t}$; this is exactly the order $\mathcal{O}'(r)$ in [Ibu82]. Now, embed $\mathcal{O}_t$ in $\mathcal{O}_0 \otimes \mathbb{Q}$ via the isomorphism from [ESPV23, Lemma 10]. Find an element γ of minimal odd relative norm in the connecting ideal $\mathcal{O}_0\mathcal{O}_t$ to compute the integral odd $(\mathcal{O}_0, \mathcal{O}_t)$-ideal $J_t := \mathcal{O}_0\mathcal{O}_t\gamma^{-1}$. Compute the associated isogeny $\varphi_{J_t} \colon E_0 \to E_t$ and the images $P_t = \varphi_{J_t}(P_0)$, $Q_t = \varphi_{J_t}(Q_0)$ using an ideal-to-isogeny algorithm such as Algorithm 3.13. The matrices $\mathbf{M}_{t,u}$ can be computed by finding an explicit isomorphism $\mathcal{O}_t \xrightarrow{\sim} \mathrm{End}(E_t)$ using the methods of [ESPV23, § 3.1], then evaluating the endomorphisms corresponding to $b_{t,u}$ on the basis (P_t, Q_t) of $E_t[2^f]$ and solving a few two-dimensional DLPs.

3.2.2. Representation of isogenies between elliptic curves via $(2,2)$-isogenies

Let d_1, d_2, e be positive integers such that $\gcd(d_1, d_2) = 1$ and $d_1 + d_2 = 2^e$. Consider the following commutative diagram of isogenies between elliptic curves:

$$\begin{array}{ccc} E_1 & \xrightarrow{\varphi_1} & E_2 \\ \downarrow{\scriptstyle \varphi_2} & \searrow^{\psi} & \downarrow{\scriptstyle \varphi_2'} \\ E_3 & \xrightarrow{\varphi_1'} & E_4 \end{array}$$

where $\deg(\varphi_1) = \deg(\varphi_1') = d_1$ and $\deg(\varphi_2) = \deg(\varphi_2') = d_2$. Then the isogeny $\Phi : E_1 \times E_4 \to E_2 \times E_3$ defined by $\begin{pmatrix} \varphi_1 & \widehat{\varphi_2'} \\ -\varphi_2 & \widehat{\varphi_1'} \end{pmatrix}$ is a $(2^e, 2^e)$-isogeny with kernel $\{([d_1]P, \psi(P)) \mid P \in E_1[2^e]\}$.

Given E_1, E_4, d_1, d_2, e, and the restriction of ψ on $E_1[2^e]$, we can compute an isogeny $\iota \circ \Phi$ by Isogeny22Chain for an isomorphism ι between products of elliptic curves. Suppose that $E_2 \neq E_3$, thus $e \geq 2$. Then, for $P \in E_1$, the first component of $\iota \circ \Phi(P, 0_{E_4})$ is $\varphi_1(P)$ or $\varphi_2(P)$ (up to post-composition with an isomorphism). We can determine which one is the case by computing the Weil pairing as follows. Let (P, Q) be a basis of $E_1[2^{e'}]$ for $e' \geq 2$ and $(P_1, P_2) = \iota \circ \Phi(P, 0_{E_4})$, $(Q_1, Q_2) = \iota \circ \Phi(Q, 0_{E_4})$. Then, we have $e_{2^{e'}}(P_1, Q_1) = e_{2^{e'}}(P, Q)^{d_1}$ if $P_1 = \varphi_1(P)$, and $e_{2^{e'}}(P_1, Q_1) = e_{2^{e'}}(P, Q)^{d_2}$ if $P_1 = \varphi_2(P)$. Since $d_1 \equiv -d_2 \pmod{2^e}$ and $f \geq 2$, we have $d_1 \not\equiv d_2 \pmod{2^{e'}}$. Therefore, we can determine which is the case.

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

Let I be a left $\mathcal{O}_0$-ideal and E_I be the codomain of φ_I. Given I, we can compute E_I and the images of P_0, Q_0 under φ_I by the following algorithm, which is summarized in IdealToIsogeny.

First, we compute positive integers u, v, e, indices s, t, and $\beta_1 \in \overline{J_s}I, \beta_2 \in \overline{J_t}I$ such that

$$ud_1 + vd_2 = 2^e, \quad \text{and} \quad \gcd(ud_1, vd_2) = 1, \quad \text{and} \quad e \leq f,$$

where $d_1 = \mathrm{nrd}(\beta_1)/\mathrm{nrd}(\overline{J_s}I), d_2 = \mathrm{nrd}(\beta_2)/\mathrm{nrd}(\overline{J_t}I)$. This is accomplished by SuitableIdeals (see Section 3.2.5).

Then, we set $I_1 = \chi_{\overline{J_s}I}(\beta_1)$ and $I_2 = \chi_{\overline{J_t}I}(\beta_2)$, and we obtain that

$$\beta_1 = \widehat{\varphi_{I_1}} \circ \varphi_I \circ \widehat{\varphi_{J_s}}, \qquad \beta_2 = \widehat{\varphi_{I_2}} \circ \varphi_I \circ \widehat{\varphi_{J_t}}.$$

This implies that $\beta_2 \circ \varphi_{J_t} \circ \widehat{\varphi_{J_s}} \circ \overline{\beta_1} = [\mathrm{nrd}(I)\,\mathrm{nrd}(J_s)\,\mathrm{nrd}(J_t)]\widehat{\varphi_{I_2}} \circ \varphi_{I_1}$, which means that we can compute the images of P_s, Q_s under $\widehat{\varphi_{I_2}} \circ \varphi_{I_1}$ by

$$\widehat{\varphi_{I_2}} \circ \varphi_{I_1} \begin{pmatrix} P_s \\ Q_s \end{pmatrix} = \left[\frac{1}{\mathrm{nrd}(I)\,\mathrm{nrd}(J_t)}\right] \mathbf{M}_{\overline{\beta_1}} \mathbf{M}_{\beta_2} \begin{pmatrix} P_t \\ Q_t \end{pmatrix}.$$

Next, we compute isogenies

$$\varphi_u : E_s \to E_u, \ \varphi_v : E_t \to E_v$$

of degrees u and v respectively, using FixedDegreeIsogeny (see Section 3.2.4), which gives us the diagram in Figure 2. As explained in Section 3.2.2, we can obtain E_I and $\varphi_{I_1} \circ \widehat{\varphi_u}$ by computing $(2^e, 2^e)$-isogeny with kernel

$$\{([ud_1]P, \varphi_v \circ \widehat{\varphi_{I_2}} \circ \varphi_{I_1} \circ \widehat{\varphi_u}(P)) \mid P \in E_u[2^e]\}.$$

Since φ_u induces a bijection from $E_s[2^e]$ to $E_u[2^e]$, the kernel is equal to

$$\{([d_1]\varphi_u(P), \varphi_v \circ \widehat{\varphi_{I_2}} \circ \varphi_{I_1}(P)) \mid P \in E_i[2^e]\}.$$

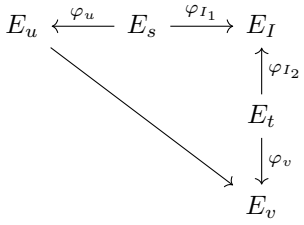

FIGURE 2. The isogenies in the algorithm IdealToIsogeny.

Finally, we obtain the images $\varphi_I(P_0), \varphi_I(Q_0)$ as

$$\varphi_I(P_0) = [1/\operatorname{nrd}(I_1)]\varphi_{I_1} \circ \beta_1(P_s), \quad \varphi_I(Q_0) = [1/\operatorname{nrd}(I_1)]\varphi_{I_1} \circ \beta_1(Q_s).$$

Algorithm 3.13 IdealToIsogeny(I)

Input: A left $\mathcal{O}_0$-ideal I.
Output: The codomain E_I of φ_I, $\varphi_I(P_0)$, and $\varphi_I(Q_0)$.

$u, v, e, s, t, \beta_1, \beta_2 \leftarrow$ SuitableIdeals(I)
$d_1 \leftarrow \operatorname{nrd}(\beta_1)/\operatorname{nrd}(\overline{J_s}I)$
$d_2 \leftarrow \operatorname{nrd}(\beta_2)/\operatorname{nrd}(\overline{J_t}I)$ // $ud_1 + vd_2 = 2^e$, with $\gcd(ud_1, vd_2) = 1$
$E_u, \varphi_u(P_s), \varphi_u(Q_s) \leftarrow$ FixedDegreeIsogeny(s, u) // $\varphi_u : E_s \to E_u$ is an isogeny of degree u
$E_v, \varphi_v(P_t), \varphi_v(Q_t) \leftarrow$ FixedDegreeIsogeny(t, v) // $\varphi_v : E_t \to E_v$ is an isogeny of degree v
$[P,\, Q]^T \leftarrow \left[\frac{1}{\operatorname{nrd}(I)\operatorname{nrd}(J_t)}\right] \mathbf{M}_{\overline{\beta_1}}\mathbf{M}_{\beta_2}[\varphi_v(P_t),\, \varphi_v(Q_t)]^T$
$K_P \leftarrow [2^{f-e}]\big([d_1]\varphi_u(P_s), P\big)$
$K_Q \leftarrow [2^{f-e}]\big([d_1]\varphi_u(Q_s), Q\big)$
$E \times E', [(P, P'), (Q, Q')] \leftarrow$ Isogeny22Chain$(K_P, K_Q, [(\varphi_u(P_s), 0_{E_v}), (\varphi_u(Q_s), 0_{E_v})])$
if $e_{2^f}(P, Q) = e_{2^f}(P_s, Q_s)^{u^2 d_1}$ **then** // We identify the right curve E_I by ensuring it is d_1-isogeneous to E_0
  $E_I \leftarrow E$, $P_I \leftarrow P$, and $Q_I \leftarrow Q$
else
  $E_I \leftarrow E'$, $P_I \leftarrow P'$, and $Q_I \leftarrow Q'$
$[P_I, Q_I]^T \leftarrow \left[\frac{1}{ud_1}\right] \mathbf{M}_{\beta_1}[P_I,\, Q_I]^T$
return $E_I, (P_I, Q_I)$

Ideals of even norm. In SQISIGN, we also need to consider ideals of even norm. Specifically, we need to handle ideals of norm $n2^e$ for an odd integer n and $e \le f$. Let I be a left $\mathcal{O}_0$-ideal of norm $n2^e$. Then we can decompose I as $I = I_1 I_2$ where I_1 is a left $\mathcal{O}_0$-ideal of norm n and I_2 is a left $\mathcal{O}_R(I_1)$-ideal of norm 2^e. By IdealToIsogeny, we can compute the codomain E_{I_1} of φ_{I_1} and the images of P_0, Q_0 under φ_{I_1}. Let $a, b \in \mathbb{Z}$ be integers such that the kernel of the isogeny corresponding to the pullback $[I_1]^* I_2$ is generated by

$$[a2^{f-e}]P_0 + [b2^{f-e}]Q_0.$$

Then the kernel of φ_{I_2} is generated by the image of this point under φ_{I_1}. The integers a, b can be computed by IdealToKernel (Algorithm 3.14).

Algorithm 3.14 IdealToKernel(I)

Input: A left $\mathcal{O}_0$-ideal I of norm 2^e for $e \le f$.
Output: Integers a, b such that $\ker \varphi_I$ is generated by $[a2^{f-e}]P_0 + [b2^{f-e}]Q_0$.

Compute $\alpha \in \mathcal{O}_0$ such that $I = \mathcal{O}_0\langle \alpha, 2^e \rangle$
Compute $[a, b]^T$ in the right kernel of $\mathbf{M}_\alpha \bmod 2^e$
return a, b

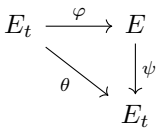

FIGURE 3. The isogenies in the algorithm FixedDegreeIsogeny.

3.2.4. Evaluating an arbitrary fixed odd degree isogeny from a precomputed curve

For an index t and a given positive odd integer u such that $u < 2^{f-2}$, we can compute the codomain E of a u-isogeny $\varphi : E_t \to E$ and the images of P_t, Q_t under φ by the following algorithm, which is depicted in FixedDegreeIsogeny.

The overview of the algorithm is as follows. Let e_{FDI} be the smallest value between $f-2$ and $\lceil \log_2(p) - \log_2(u) \rceil +$ `QUAT_repres_bound_input`. First, we compute an endomorphism θ of E_i of norm $u(2^{e_{\text{FDI}}} - u)$ by GeneralizedRepresentInteger. If the computation succeeds, then the returned endomorphism θ factors as $\theta = \varphi \circ \psi$ for some u-isogeny $\varphi : E_t \to E$ and $(2^{e_{\text{FDI}}} - u)$-isogeny $\psi : E \to E_t$. Next, we compute the $(2^{e_{\text{FDI}}}, 2^{e_{\text{FDI}}})$-isogeny with kernel $\{([u]P, \theta(P)) \mid P \in E_t[2^{e_{\text{FDI}}}]\}$ and obtain the codomain E of φ and the images of P_t, Q_t under φ. The diagram in Figure 3 shows the isogenies in the algorithm FixedDegreeIsogeny.

Algorithm 3.15 FixedDegreeIsogeny(t, u)

Input: An index t and a positive odd integer $u < 2^{f-2}$.
Output: The curve E, $\varphi(P_t)$, and $\varphi(Q_t)$, where $\varphi : E_t \to E$ is a u-isogeny.

1: $e_{\text{FDI}} \leftarrow \min(f-2, \lceil \log_2(p) - \log_2(u) \rceil +$ `QUAT_repres_bound_input`$)$
2: $\theta \leftarrow$ GeneralizedRepresentInteger$(u(2^{e_{\text{FDI}}} - u), \sqrt{-q_t}, \mathcal{O}_t,$ `true`$)$
3: $[\theta(P_t), \theta(Q_t)]^T \leftarrow \mathbf{M}_\theta [P_t, Q_t]^T$
4: $K_1 \leftarrow [2^{f-2-e_{\text{FDI}}}]([u]P_t, \theta(P_t))$
5: $K_2 \leftarrow [2^{f-2-e_{\text{FDI}}}]([u]Q_t, \theta(Q_t))$
6: $E \times E', [(P, P'), (Q, Q')] \leftarrow$ Isogeny22ChainWithTorsion$(K_1, K_2, [(P_t, 0_{E_t}), (Q_t, 0_{E_t})])$
7: **if** $e_{2^f}(P, Q) \neq e_{2^f}(P_t, Q_t)^u$ **then**
8: $\quad E \leftarrow E'$, $P \leftarrow P'$, and $Q \leftarrow Q'$
9: **return** E, P, Q

We remark that for Algorithm 3.15, since we are using the isogeny algorithm Isogeny22ChainWithTorsion that takes extra torsion as input, then using the implementation of Algorithm 8.48 the codomain at the end is set up in such a way that E is always the correct curve in line 6, and the pairing check in line 7–8 can be skipped.

3.2.5. Finding suitable ideals

The algorithm SuitableIdeals finds positive integers u, v, e, indices s, t, and ideals I_1, I_2 in the diagram in Figure 2. The outline of the algorithm is as follows. First, we find $\beta_s \in \overline{J_s} I$ and $\beta_t \in \overline{J_t} I$ such that $d_1 = \operatorname{nrd}(\beta_1)/\operatorname{nrd}(\overline{J_s} I)$ and $d_2 = \operatorname{nrd}(\beta_2)/\operatorname{nrd}(\overline{J_t} I)$ are odd, coprime, and satisfy $d_1, d_2 < 2^f$. Next, we compute integers u and v such that $ud_1 + vd_2 = 2^f$. This can be achieved by setting $u \leftarrow 2^f d_1^{-1} \pmod{d_2}$ and $v \leftarrow (2^f - ud_1)/d_2$. If $v > 0$, the dyadic valuations of u and v are equal, and u and v do not share any odd common divisor, as $ud_1 + vd_2 = 2^f$. Consequently, the greatest odd divisors of u and v satisfy the required conditions.

3.2.6. 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.

Algorithm 3.16 SuitableIdeals(I)

Input: A left $\mathcal{O}_0$-ideal I.
Output: Positive integers u, v, e, indices s, t, and $\beta_1 \in \overline{J_s}I, \beta_2 \in \overline{J_t}I$ such that $ud_1 + vd_2 = 2^e$, $\gcd(ud_1, vd_2) = 1$, and $e \leq f$, where $d_1 = \mathrm{nrd}(\beta_1)/\mathrm{nrd}(\overline{J_s}I), d_2 = \mathrm{nrd}(\beta_2)/\mathrm{nrd}(\overline{J_t}I)$.

for t **from** 0 **up to** n_{orders} **do**
  Let $(b_1, \ldots, b_4)$ be a basis of $\overline{J_t}I$
  Compute the Gram matrix $\mathbf{G} = (\langle b_r, b_s \rangle)_{1 \leq r,s \leq 4}$
  $(\alpha_1, \ldots, \alpha_4), \_ \leftarrow \mathsf{L2}_{\delta,\eta}((b_1, \ldots, b_4), \mathbf{G})$
  $L_t \leftarrow \left[\sum_{s=1}^{4} x_s \alpha_s \;\middle|\; (x_1, \ldots, x_4) \in [-m, \ldots, m] \right]$ for $m =$ `FINDUV_box_size`
  Sort L_t by increasing norm.
for $(s, t) \in [0, \ldots, n_{\text{orders}}]^2$ **do** // by increasing lexicographic order
  for $(\beta_1, \beta_2) \in L_s \times L_t$ **do** // by increasing lexicographic order
    $d_1 \leftarrow \mathrm{nrd}(\beta_1)/\mathrm{nrd}(\overline{J_s}I)$ and $d_2 \leftarrow \mathrm{nrd}(\beta_2)/\mathrm{nrd}(\overline{J_t}I)$
    if $d_1 \equiv 1 \pmod 2$ and $d_2 \equiv 1 \pmod 2$ and $\gcd(d_1, d_2) = 1$ **then**
      $u \leftarrow 2^f d_1^{-1} \pmod{d_2}$
      $v \leftarrow (2^f - ud_1)/d_2$
      if $v > 0$ **then**
        $e \leftarrow \mathsf{DyadicValuation}(u)$
        $u \leftarrow u/2^e$, $v \leftarrow v/2^e$, and $e \leftarrow f - e$
        return $u, v, e, s, t, \beta_1, \beta_2$
raise Exception("SuitableIdeals failed")

Using a decomposition along a fixed basis. In SQIsign, we only ever translate points on E_0 of order 2^f 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 3.17) takes in the decomposition of P along the fixed precomputed basis (P_0, Q_0), i.e., $P = [a]P_0 + [b]Q_0$. By having again precomputed the action of θ and η on this basis, we obtain the required ideal without requiring any discrete log computations.

Algorithm 3.17 KernelDecomposedToIdeal$_D(c_1, c_2)$

Input: $c_1, c_2 \in \mathbb{Z}$ defining a point $[c_1]P_0 + [c_2]Q_0$ on E_0 of order 2^f generating an isogeny φ.
Output: I_φ, a left $\mathcal{O}_0$-ideal.

$[d_1, d_2]^T \leftarrow \mathbf{M}_\theta [c_1, c_2]^T$
$\mathbf{M} \leftarrow \begin{pmatrix} c_1 & d_1 \\ c_2 & d_2 \end{pmatrix}$
$[a, b]^T \leftarrow \mathbf{M}^{-1} \mathbf{M}_i [c_1, c_2]^T$
$I \leftarrow \mathcal{O}_0 \left\langle a + b\left(j + \frac{1+k}{2}\right) - i, 2^f \right\rangle$
return I

CHAPTER 4

The Signature Scheme

This chapter provides a detailed explanation of the key generation, signing, and verification procedures in SQISIGN.

4.1. Σ protocols and the Fiat–Shamir Transform

The signature scheme SQISIGN is constructed from a Σ protocol using the Fiat–Shamir transform. For completeness, the relevant definitions are recalled. Such notions will be used to prove the security of SQISIGN in Chapter 10.

Σ protocols. To begin, the definition of Σ protocol is introduced. In the context of SQISIGN, Σ protocols will always be *commitment recoverable*. Hence, the definitions below are tailored to this case.

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 4.1.1 (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 $\mathsf{chl} \leftarrow \{0,1\}^\lambda$ and sends the challenge to the prover.
(3) On receiving the challenge chl from the verifier, the prover runs $\mathsf{resp} \leftarrow \mathsf{P}_2(\mathsf{chl})$ and returns the response resp to the verifier.
(4) The verifier runs the verification algorithm $\mathsf{V}(x, \mathsf{chl}, \mathsf{resp})$ and outputs com'.

We refer to the tuple $(\mathsf{com}, \mathsf{chl}, \mathsf{resp})$ as a **transcript** of the Σ protocol. A transcript $(\mathsf{com}, \mathsf{chl}, \mathsf{resp})$ is said to be valid, or accepting, if $\mathsf{V}(x, \mathsf{chl}, \mathsf{resp}) = \mathsf{com}$. 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 the commitment chosen by the prover; (2) special soundness, if a cheating prover only knows statement x and not a corresponding witness w, they cannot force an honest verifier to output a valid commitment; (3) honest-verifier zero-knowledge, no malicious verifier cannot extract additional knowledge from the prover. These are defined in more detail in Chapter 10.

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 secret key $\mathsf{sk} = (x, w)$ and the public key $\mathsf{pk} = x$.
- $\mathsf{Sign}(\mathsf{sk}, \mathsf{msg})$: On input the secret key $\mathsf{sk} = (x, w)$ and a message $\mathsf{msg} \in \{0,1\}^*$, sample

$$\mathsf{com} \leftarrow \mathsf{P}_1(x, w), \quad \mathsf{chl} = H(\mathsf{com} \| \mathsf{msg}), \quad \mathsf{resp} = \mathsf{P}_2(\mathsf{chl}),$$

  and output the signature $\sigma = (\mathsf{chl}, \mathsf{resp})$.
- $\mathsf{Ver}(\mathsf{pk}, \mathsf{msg}, \sigma)$: On input the public key $\mathsf{pk} = x$, a message $\mathsf{msg} \in \{0,1\}^*$ and a signature $\sigma = (\mathsf{chl}, \mathsf{resp})$, compute $\mathsf{com}' = \mathsf{V}(x, \mathsf{chl}, \mathsf{resp})$. Return `true` if $\mathsf{chl} = H(\mathsf{com}' \| \mathsf{msg})$, else output `false`.

4.2. Parameters

The SQIsign algorithm works with several parameters and precomputed values: all algorithms implicitly take a parameter set as input. For the specific values of such parameters, we refer the reader to Chapter 5. Hereafter, we provide a list of these parameters, and in some cases also describe how they can be derived from other parameters. Generally, the values and parameters can be divided in two categories: the parameters directly related to the algorithms of SQIsign, and those that are related to some algorithmic building blocks. The former are treated in Section 4.2.1, and the latter in Section 4.2.2. We assume that all the algorithms described below have access to the parameter values.

4.2.1. Scheme parameters

The parameters and precomputed values of SQIsign include:

- The prime $p = c \cdot 2^f - 1$, where c is a small positive integer and $f \approx 2\lambda$.
- The degree D_{mix}, used by the secret isogeny and the commitment isogeny, to guarantee stationary distribution of the codomain curve. Concretely, D_{mix} is chosen as the smallest prime larger than $2^{4\lambda}$.
- The upper bound D_{rsp} for the degree of the response isogeny; for convenience, we also define e_{rsp} for its 2-logarithm, i.e., $D_{\text{rsp}} = 2^{e_{\text{rsp}}}$; concretely $e_{\text{rsp}} = \left\lceil \log_2\left(\sqrt{p}\right)\right\rceil$.
- The degree D_{chl} of the challenge isogeny., which is defined as $D_{\text{chl}} = 2^f$.
- The exponent $e_{\text{chl}} = f - e_{\text{rsp}}$ defining the challenge space: challenges are integers $0 < \mathsf{chl} < 2^{e_{\text{chl}}}$.
- A basis $B_0 = (P_0, Q_0)$ of $E_0[2^f]$, deterministically generated by $\mathsf{TorsionBasis}_0(\mathbb{F}_{p^2})$.
- A hash function HASH, obtained by repeating SHAKE256 multiple times.

4.2.2. Algorithmic parameters

Several algorithmic building blocks (mostly algorithms of Chapter 3) used in SQIsign require some constants. The choice of these constants can be critical in some cases as it determines the failure probability of these algorithms. In particular, we will explain how to choose those constants to ensure that the failure probability is smaller than 2^{-64}. Some justifications regarding these choices can be found in Chapter 9.

- `QUAT_primality_num_iter` is the number of Miller-Rabin tests used in to test pseudo-primality for the equivalent ideal functionality. It should be such that $4^{-\texttt{QUAT\_primality\_num\_iter}} < 2^{-64}$ (this is a common upper-bound on the probability of failure of repetitions of Miller Rabin test).
- FixedDegreeIsogeny calls to GeneralizedRepresentInteger with an input $M = u(2^{e_{\text{FDI}}} - u)$. The exponent e_{FDI} is chosen to ensure that the execution of GeneralizedRepresentInteger succeeds with good probability. For this reason e_{FDI} is computed from a constant `QUAT_repres_bound_input` computed as follows:
$$\texttt{QUAT\_repres\_bound\_input} = \left\lceil \log_2\left(\log_{1-1/(64\cdot\log_2(p))}(2^{-64})\right)\right\rceil$$
- The bound `QUAT_equiv_bound_coeff` is computed as $2^{1+\lfloor \texttt{QUAT\_repres\_bound\_input}/4\rfloor}$
- The bound `FINDUV_box_size` is computed as $2 + \lfloor(\log_2(p) - \mathsf{DyadicValuation}(p+1))/4\rfloor$.
- The number n_{orders} of alternate orders and the associated list of integers q_t defining them are set for each security level as follows:
  NIST-I: $n_{\text{orders}} = 6, \quad (q_t) = (5, 17, 37, 41, 53, 97)$;
  NIST-III: $n_{\text{orders}} = 7, \quad (q_t) = (5, 13, 17, 41, 73, 89, 97)$;
  NIST-V: $n_{\text{orders}} = 6, \quad (q_t) = (5, 37, 61, 97, 113, 149)$.

4.3. Key generation

The key generation algorithm SQIsign.KeyGen (Algorithm 4.1) produces a public key pk, which is a random curve E_{pk} and some basis hint $\mathsf{hint}_{\mathsf{pk}}$, and a secret key sk, which is a representation of a secret isogeny $\varphi_{\mathsf{sk}} : E_0 \to E_{\mathsf{pk}}$. This isogeny can also be interpreted as a representation of the endomorphism ring $\mathrm{End}(E_{\mathsf{pk}})$ of E_{pk}.

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

(1) The public key curve E_{pk}.

(2) The hint $\mathsf{hint}_{\mathsf{pk}}$ used to generate a deterministic basis on E_{pk}.
(3) The secret ideal I_{sk} corresponding to the secret isogeny $\varphi_{\mathsf{sk}} : E_0 \to E_{\mathsf{pk}}$.
(4) The change-of-basis matrix $\mathbf{M}_{\mathsf{sk}}$ from $(\varphi_{\mathsf{sk}}(P_0), \varphi_{\mathsf{sk}}(Q_0))$ to B_{pk}.

More specifically, the key generation proceeds by sampling a random secret left $\mathcal{O}_0$-ideal I_{sk} of prime norm D_{mix} via RandomIdealGivenNorm. To obtain an ideal of smaller norm, the ideal I_{sk} is updated to be the output of the algorithm RandomEquivalentPrimeIdeal. Then, using IdealToIsogeny, the corresponding secret isogeny $\varphi_{\mathsf{sk}} : E_0 \to E_{\mathsf{pk}}$ is evaluated on the 2^f-torsion to obtain the points $\varphi_{\mathsf{sk}}(P_0)$ and $\varphi_{\mathsf{sk}}(Q_0)$. Since RandomEquivalentPrimeIdeal and IdealToIsogeny may fail, it might be necessary to rerun them on a different input: if any of the three algorithms fails, a new secret ideal I_{sk} is sampled using fresh randomness.

To avoid recomputing φ_{sk} during SQIsign.Sign, the information $(\varphi_{\mathsf{sk}}(P_0), \varphi_{\mathsf{sk}}(Q_0))$ is stored in the secret key sk. We now briefly explain how this is done. First, we generate a deterministic basis $B_{\mathsf{pk}} = (P_{\mathsf{pk}}, Q_{\mathsf{pk}})$ of $E_{\mathsf{pk}}[2^f]$. Then, we compute the change-of-basis matrix $\mathbf{M}_{\mathsf{sk}} = \left(\begin{smallmatrix} a & b \\ c & d \end{smallmatrix}\right)$ such that

$$\mathbf{M}_{\mathsf{sk}} \cdot \begin{pmatrix} \varphi_{\mathsf{sk}}(P_0) \\ \varphi_{\mathsf{sk}}(Q_0) \end{pmatrix} = \begin{pmatrix} [a]\varphi_{\mathsf{sk}}(P_0) + [b]\varphi_{\mathsf{sk}}(Q_0) \\ [c]\varphi_{\mathsf{sk}}(P_0) + [d]\varphi_{\mathsf{sk}}(Q_0) \end{pmatrix} = \begin{pmatrix} P_{\mathsf{pk}} \\ Q_{\mathsf{pk}} \end{pmatrix}.$$

The matrix $\mathbf{M}_{\mathsf{sk}}$ is part of the secret key sk. Also, as we generate the deterministic basis B_{pk}, we save a hint $\mathsf{hint}_{\mathsf{pk}}$, which we append to the public key pk. The hint $\mathsf{hint}_{\mathsf{pk}}$ is then used to recompute B_{pk} faster during SQIsign.Verify (Algorithm 4.9). We refer to Section 2.2.3 for more details about the generation of torsion bases with hints.

Algorithm 4.1 SQIsign.KeyGen( )

Output: Secret key `sk` and public key `pk`.

while `true` **do**
  $I_{\mathsf{sk}} \leftarrow \mathsf{RandomIdealGivenNorm}(D_{\mathsf{mix}}, \texttt{true})$
  try
    $I_{\mathsf{sk}} \leftarrow \mathsf{RandomEquivalentPrimeIdeal}(I_{\mathsf{sk}})$
    $E_{\mathsf{pk}}, \varphi_{\mathsf{sk}}(P_0), \varphi_{\mathsf{sk}}(Q_0) \leftarrow \mathsf{IdealToIsogeny}(I_{\mathsf{sk}})$
  except
    continue
  $P_{\mathsf{pk}}, Q_{\mathsf{pk}}, \mathsf{hint}_{\mathsf{pk}} \leftarrow \mathsf{TorsionBasisToHint}(E_{\mathsf{pk}})$
  $\mathbf{M}_{\mathsf{sk}} \leftarrow \mathsf{ChangeOfBasis}_{2^f}(E_{\mathsf{pk}}, (\varphi_{\mathsf{sk}}(P_0), \varphi_{\mathsf{sk}}(Q_0)), (P_{\mathsf{pk}}, Q_{\mathsf{pk}}))$
  $\mathsf{pk} \leftarrow (E_{\mathsf{pk}}, \mathsf{hint}_{\mathsf{pk}})$
  $\mathsf{sk} \leftarrow (E_{\mathsf{pk}}, \mathsf{hint}_{\mathsf{pk}}, I_{\mathsf{sk}}, \mathbf{M}_{\mathsf{sk}})$
  return sk, pk

4.4. Signing

Following the framework in Section 4.1, the SQIsign signing algorithm can naturally be divided into three stages: commitment phase, challenge phase, and the response phase. This signing procedure SQIsign.Sign (Algorithm 4.2) takes as input a secret key `sk` and message msg. At a high level, the commitment phase consists of computing a random isogeny $\varphi_{\mathsf{com}} : E_0 \to E_{\mathsf{com}}$ and the ideal I_{com}. During the challenge phase, the commitment E_{com} and the message msg are hashed to generate a challenge chl. Such a challenge defines a challenge isogeny $\varphi_{\mathsf{chl}} : E_{\mathsf{pk}} \to E_{\mathsf{chl}}$ whose corresponding ideal is denoted by I_{chl}. Finally, the response phase consists of computing an equivalent ideal $I_{\mathsf{rsp}} \sim \overline{I_{\mathsf{com}}} \cdot I_{\mathsf{sk}} \cdot I_{\mathsf{chl}}$ and translating I_{rsp} to its corresponding isogeny $\varphi_{\mathsf{rsp}} : E_{\mathsf{com}} \to E_{\mathsf{chl}}$. The output of the signing procedure is a signature σ containing the challenge chl and some additional data describing the response isogeny $\varphi_{\mathsf{rsp}} : E_{\mathsf{com}} \to E_{\mathsf{chl}}$. Below, we give more detail on each phase of the signing procedure.

4.4.1. Commitment

The commitment starts by sampling a random commitment ideal I_{com} of norm D_{mix} via RandomIdealGivenNorm. Such an ideal is then randomized via RandomEquivalentPrimeIdeal. Then, the isogeny $\varphi_{\mathsf{com}} : E_0 \to E_{\mathsf{com}}$ corresponding to ideal I_{com} is computed using IdealToIsogeny. Additionally, the isogeny $\varphi_{\mathsf{com}} : E_0 \to E_{\mathsf{com}}$ is evaluated on the basis $B_0 = (P_0, Q_0)$. Analogously to SQIsign.KeyGen, RandomEquivalentPrimeIdeal and IdealToIsogeny may fail. Hence, it might be necessary to restart the signing procedure using a different ideal, sampled using new randomness.

Algorithm 4.2 SQIsign.Sign(sk, msg)

Input: Secret signing key sk and message msg $\in \{0,1\}^*$.
Output: Signature σ.

Parse sk as $(E_{\mathsf{pk}}, \mathsf{hint}_{\mathsf{pk}}, I_{\mathsf{sk}}, \mathbf{M}_{\mathsf{sk}})$
$P_{\mathsf{pk}}, Q_{\mathsf{pk}} \leftarrow \mathsf{TorsionBasisFromHint}(E_{\mathsf{pk}}, \mathsf{hint}_{\mathsf{pk}})$
while `true` **do**
  // **Commitment**
  $I_{\mathrm{com}} \leftarrow \mathsf{RandomIdealGivenNorm}(D_{\mathsf{mix}}, \mathtt{true})$
  try
    $I_{\mathrm{com}} \leftarrow \mathsf{RandomEquivalentPrimeIdeal}(I_{\mathrm{com}})$
    $E_{\mathsf{com}}, P_{\mathsf{com}}, Q_{\mathsf{com}} \leftarrow \mathsf{IdealToIsogeny}(I_{\mathrm{com}})$
  except
    continue
  // **Challenge**
  $\mathsf{chl} \leftarrow \mathsf{HASH}(\mathsf{pk} \,||\, j(E_{\mathsf{com}}) \,||\, \mathsf{msg})$ // See Section 4.6 for the binary encoding of elements in $\mathbb{F}_{p^2}$
  // **Response**
  $(c_1, c_2) \leftarrow \mathbf{M}_{\mathsf{sk}} \cdot (1, \mathsf{chl})$
  $I'_{\mathrm{chl}} \leftarrow \mathsf{KernelDecomposedToIdeal}_{2^f}(c_1, c_2)$
  $I_{\mathrm{chl}} \leftarrow [I_{\mathsf{sk}}]_* I'_{\mathrm{chl}}$ // I_{chl} is the ideal corresponding to $\varphi_{I_{\mathrm{chl}}}$
  $\alpha_{\mathrm{rsp}} \leftarrow \mathsf{RandomEquivalentQuaternion}(\overline{I_{\mathrm{com}}} \cap I_{\mathsf{sk}} \cdot I_{\mathrm{chl}})$
  $\alpha_{\mathrm{rsp}}, n_{\mathrm{bt}} \leftarrow \mathsf{ComputeBacktrackingAndNormalize}(\alpha_{\mathrm{rsp}})$
  $d_{\mathrm{rsp}} \leftarrow \mathrm{nrd}(\alpha_{\mathrm{rsp}})/D_{\mathsf{mix}}^2 \cdot 2^{f-n_{\mathrm{bt}}}$ // d_{rsp} is the degree of φ_{rsp}
  $r_{\mathrm{rsp}} \leftarrow \mathsf{DyadicValuation}(d_{\mathrm{rsp}})$
  $q_{\mathrm{rsp}} \leftarrow d_{\mathrm{rsp}}/2^{r_{\mathrm{rsp}}}$
  $I_{\mathrm{com,rsp}} \leftarrow \mathcal{O}_0\alpha_{\mathrm{rsp}} + \mathcal{O}_0(q_{\mathrm{rsp}} D_{\mathsf{mix}})$ // $I_{\mathrm{com,rsp}}$ is the ideal corresponding to $\varphi_{\mathrm{com}} \circ \varphi_{\mathrm{rsp}}$
  $e'_{\mathrm{rsp}} \leftarrow e_{\mathrm{rsp}} - r_{\mathrm{rsp}} - n_{\mathrm{bt}}$ // e'_{rsp} is the degree of the two-dimensional isogeny
  if $e'_{\mathrm{rsp}} > 0$ **then**
    try
      $I_{\mathrm{aux}} \leftarrow \mathsf{RandomIdealGivenNorm}(2^{e'_{\mathrm{rsp}}} - q_{\mathrm{rsp}}, \mathtt{false})$
      $E'_{\mathrm{aux}}, P'_{\mathrm{aux}}, Q'_{\mathrm{aux}} \leftarrow \mathsf{IdealToIsogeny}(I_{\mathrm{com,rsp}} \cap I_{\mathrm{aux}})$
    except
      continue
    $E_{\mathrm{aux}}, P_{\mathrm{aux}}, Q_{\mathrm{aux}}, E_{\mathsf{chl}}, P_{\mathsf{chl}}, Q_{\mathsf{chl}} \leftarrow \mathsf{SplitAuxiliaryIsogeny}(E_{\mathsf{com}}, E'_{\mathrm{aux}}, P_{\mathsf{com}}, Q_{\mathsf{com}}, P'_{\mathrm{aux}}, Q'_{\mathrm{aux}}, q_{\mathrm{rsp}}, e'_{\mathrm{rsp}}, r_{\mathrm{rsp}})$
  else
    try
      $E_{\mathsf{chl}}, P_{\mathsf{chl}}, Q_{\mathsf{chl}} \leftarrow \mathsf{IdealToIsogeny}(I_{\mathrm{com,rsp}})$
      $E_{\mathrm{aux}}, P_{\mathrm{aux}}, Q_{\mathrm{aux}} \leftarrow E_{\mathsf{chl}}, P_{\mathsf{chl}}, Q_{\mathsf{chl}}$
    except
      continue
  if $r_{\mathrm{rsp}} > 0$ **then**
    $E_{\mathsf{chl}}, P_{\mathsf{chl}}, Q_{\mathsf{chl}} \leftarrow \mathsf{ComputeEvenNonBacktrackingResponse}(E_{\mathsf{chl}}, P_{\mathsf{chl}}, Q_{\mathsf{chl}}, \alpha_{\mathrm{rsp}}, e'_{\mathrm{rsp}}, r_{\mathrm{rsp}})$
  $E_{\mathsf{chl}}, P_{\mathsf{chl}}, Q_{\mathsf{chl}} \leftarrow \mathsf{ComputeChallengeIsogeny}(E_{\mathsf{pk}}, \mathsf{chl}, P_{\mathsf{pk}}, Q_{\mathsf{pk}}, E_{\mathsf{chl}}, P_{\mathsf{chl}}, Q_{\mathsf{chl}}, n_{\mathrm{bt}})$
  $\mathbf{M}_{\mathsf{chl}}, \mathsf{hint}_{\mathrm{aux}}, \mathsf{hint}_{\mathsf{chl}} \leftarrow \mathsf{SetChangeOfBasisMatrix}(E_{\mathrm{aux}}, E_{\mathsf{chl}}, P_{\mathrm{aux}}, Q_{\mathrm{aux}}, P_{\mathsf{chl}}, Q_{\mathsf{chl}}, e'_{\mathrm{rsp}} + r_{\mathrm{rsp}})$
  $\sigma \leftarrow (E_{\mathrm{aux}}, n_{\mathrm{bt}}, r_{\mathrm{rsp}}, \mathbf{M}_{\mathsf{chl}}, \mathsf{chl}, \mathsf{hint}_{\mathrm{aux}}, \mathsf{hint}_{\mathsf{chl}})$
  return σ

4.4.2. Challenge

The challenge phase consists of hashing the public key pk, the j-invariant of the commitment curve E_{com}, and the message msg to a e_{chl}-bit number. Since e_{chl} is slightly less than λ, the challenge space may be too small to guarantee security when the hash function is, for instance, SHAKE256. To overcome the issue, the hash function HASH is defined as multiple iterations of SHAKE256 (this technique is sometimes known in the literature as 'grinding'). The number of iterations depends on the security level. We refer to Section 10.2.5 for more details.

4.4.3. Response

The first step during the response generation is to convert the challenge chl into an ideal I_{chl} corresponding to an isogeny originating from the curve E_{pk}. To be more specific, the ideal I_{chl} corresponds to the isogeny with kernel $\langle P_{\mathsf{pk}} + [\mathsf{chl}]Q_{\mathsf{pk}}\rangle$, where $(P_{\mathsf{pk}}, Q_{\mathsf{pk}})$ is the deterministic basis corresponding to $\mathsf{hint}_{\mathsf{pk}}$; it is computed as follows. First, we look at the pullback of the isogeny with kernel $\langle P_{\mathsf{pk}} + [\mathsf{chl}]Q_{\mathsf{pk}}\rangle$ under the secret isogeny φ_{sk}. By definition of $\mathbf{M}_{\mathsf{sk}}$, its kernel is equal to $\langle [c_1]P_0 + [c_2]Q_0\rangle$, where $(c_1, c_2) = \mathbf{M}_{\mathsf{sk}} \cdot (1, \mathsf{chl})$. We then generate the ideal I'_{chl} via the function KernelDecomposedToIdeal$_{2^f}$ and finally compute $I_{\mathsf{chl}} = [I_{\mathsf{sk}}]_* I'_{\mathsf{chl}}$.

Once we have the ideal I_{chl}, we sample a quaternion element α_{rsp} from the lattice $\overline{I_{\mathsf{com}}} \cap I_{\mathsf{sk}} \cdot I_{\mathsf{chl}}$ so that $\mathrm{nrd}(\alpha) < D_{\mathrm{rsp}} \cdot D_{\mathsf{mix}}^2 \cdot 2^f$. Under the Deuring correspondence, the lattice $\overline{I_{\mathsf{com}}} \cap I_{\mathsf{sk}} \cdot I_{\mathsf{chl}}$ can be seen as the set of all the endomorphisms α of E_{com} such that there exists an isogeny $\varphi : E_{\mathsf{com}} \to E_{\mathsf{chl}}$ verifying $\widehat{\varphi} \circ \varphi_{I_{\mathsf{chl}}} \circ \varphi_{\mathsf{sk}} \circ \widehat{\varphi}_{\mathsf{com}} = \alpha$. Hence, the element α_{rsp} can be used to describe an ideal equivalent to $J = \overline{I_{\mathsf{com}}} \cdot I_{\mathsf{sk}} \cdot I_{\mathsf{chl}}$ (specifically, the ideal $\chi_J(\alpha_{\mathrm{rsp}})$).

The element α_{rsp} must be uniformly sampled from the intersection of $\overline{I_{\mathsf{com}}} \cap (I_{\mathsf{sk}} \cdot I_{\mathsf{chl}})$ and the ball with radius $D_{\mathrm{rsp}} \cdot D_{\mathsf{mix}}^2 \cdot 2^f$. As explained in Section 3.1.5.1, the quaternion algebra $B_{p,\infty}$ is a quadratic space whose bilinear form is given by $\langle \alpha, \beta\rangle = \mathrm{tr}(\alpha\bar{\beta})$. In particular, we have $\langle \alpha, \alpha\rangle = 2\,\mathrm{nrd}(\alpha)$. Hence, to sample α_{rsp}, it use possible to use LatticeSampling with bound the quantity $D_{\mathrm{rsp}} \cdot D_{\mathsf{mix}}^2 \cdot 2^{f+1}$. The procedure is summarized in RandomEquivalentQuaternion.

Algorithm 4.3 RandomEquivalentQuaternion(L)

Input: A lattice L.
Output: A uniformly sampled $\mathbf{b} \in L$ such that $\mathrm{nrd}(\mathbf{b}) < D_{\mathrm{rsp}} \cdot D_{\mathsf{mix}}^2 \cdot 2^f$.

1: Let $(\mathbf{b}_0, \mathbf{b}_1, \mathbf{b}_2, \mathbf{b}_3)$ be a basis of L
2: Compute the Gram matrix $\mathbf{G}$ of $(\mathbf{b}_0, \mathbf{b}_1, \mathbf{b}_2, \mathbf{b}_3)$
3: $\mathbf{b} \leftarrow$ LatticeSampling$((\mathbf{b}_0, \mathbf{b}_1, \mathbf{b}_2, \mathbf{b}_3), \mathbf{G}, D_{\mathrm{rsp}} \cdot D_{\mathsf{mix}}^2 \cdot 2^{f+1})$
4: **return** $\mathbf{b}$

Let $\varphi_{\alpha_{\mathrm{rsp}}} : E_{\mathsf{com}} \to \tilde{E}$ denote the isogeny corresponding to the ideal described by α_{rsp}. The codomain of the isogeny $\varphi_{I_{\mathsf{chl}}} : E_{\mathsf{pk}} \to \tilde{E}$, corresponding to the ideal I_{chl}, coincides with that of $\varphi_{\alpha_{\mathrm{rsp}}}$. However, depending on α_{rsp}, the isogeny $\varphi_{\alpha_{\mathrm{rsp}}}$ and the isogeny $\varphi_{I_{\mathsf{chl}}}$ may share a common factor, i.e., there exist three isogenies $\psi_{\mathsf{chl}} : E_{\mathsf{pk}} \to E$, $\psi_{\mathsf{resp}} : E_{\mathsf{com}} \to E$ and $\tilde{\psi} : E \to \tilde{E}$ such that $\varphi_{\alpha_{\mathrm{rsp}}} = \tilde{\psi} \circ \psi_{\mathsf{resp}}$ and $\varphi_{I_{\mathsf{chl}}} = \tilde{\psi} \circ \psi_{\mathsf{chl}}$. Since the degree of $\varphi_{I_{\mathsf{chl}}}$ equals 2^f, the degree of the common factor is equal to $2^{n_{\mathrm{bt}}}$, where $0 \le n_{\mathrm{bt}} \le f$. Here, the notation n_{bt} refers to the fact that this quantity describes the number of steps to *backtrack* in order to have two isogenies that do not share any common factor.

In practice, the exponent n_{bt} is computed by looking at the shape of the quaternion element α_{rsp}: as mentioned above, the element α_{rsp} represents an endomorphism verifying $\widehat{\varphi} \circ \varphi_{I_{\mathsf{chl}}} \circ \varphi_{\mathsf{sk}} \circ \widehat{\varphi}_{\mathsf{com}} = \alpha_{\mathrm{rsp}}$ for some isogeny $\varphi : E_{\mathsf{com}} \to E_{\mathsf{chl}}$. Equivalently, the endomorphism α_{rsp} can be seen as an endomorphism on E_0: it is sufficient to consider the endomorphism $\widehat{\varphi}_{\mathsf{com}} \circ \widehat{\varphi} \circ \varphi_{I_{\mathsf{chl}}} \circ \varphi_{\mathsf{sk}}$. As a result, the quaternion α_{rsp} can be expressed in terms of the basis of $\mathcal{O}_0$. If we set $\alpha'_0, \alpha'_1, \alpha'_2, \alpha'_3 \in \mathbb{Z}$ such that

$$\alpha_{\mathrm{rsp}} = \alpha'_0 + \alpha'_1 i + \alpha'_2 \frac{i+j}{2} + \alpha'_3 \frac{1+k}{2},$$

then the exponent n_{bt} is the largest positive integer such that $2^{n_{\mathrm{bt}}}$ divides $\gcd(\alpha'_0, \alpha'_1, \alpha'_2, \alpha'_3)$.

Once the exponent n_{bt} has been computed, the quaternion α_{rsp} is updated to α_{rsp}/g. By doing this, the portion of backtracking between the aforementioned isogenies $\varphi_{\alpha_{\mathrm{rsp}}}$ and $\varphi_{I_{\mathsf{chl}}}$ is removed, and the isogeny described by the new α_{rsp} is now cyclic; this procedure is summarized in ComputeBacktrackingAndNormalize. The ideal described by the element α_{rsp}, which is the ideal corresponding to the response isogeny $\varphi_{\mathrm{rsp}} : E_{\mathsf{com}} \to E$, has now norm $d_{\mathrm{rsp}} = \mathrm{nrd}(\alpha_{\mathrm{rsp}})/(D_{\mathsf{mix}}^2 \cdot 2^{f-n_{\mathrm{bt}}})$.

The next step is to construct an efficient representation of φ_{rsp} to send over to the verifier. The isogeny φ_{rsp} can be seen as $\varphi_{\mathrm{rsp}} = \varphi_{\mathrm{rsp}}^{\mathrm{even}} \circ \varphi_{\mathrm{rsp}}^{\mathrm{odd}} : E_{\mathsf{com}} \to E_{\mathsf{chl}} \to E$, where $\deg(\varphi_{\mathrm{rsp}}^{\mathrm{odd}}) = q_{\mathrm{rsp}}$ is odd and $\deg(\varphi_{\mathrm{rsp}}^{\mathrm{even}}) = 2^{r_{\mathrm{rsp}}}$. The isogeny $\varphi_{\mathrm{rsp}}^{\mathrm{odd}}$ is then embedded into a two-dimensional isogeny between elliptic products, whereas the isogeny $\varphi_{\mathrm{rsp}}^{\mathrm{even}}$ is represented via its kernel.

If $e'_{\mathrm{rsp}} = e_{\mathrm{rsp}} - r_{\mathrm{rsp}} - n_{\mathrm{bt}} > 0$, the isogeny $\varphi_{\mathrm{rsp}}^{\mathrm{odd}}$ is embedded into a $(2^{e'_{\mathrm{rsp}}}, 2^{e'_{\mathrm{rsp}}})$-isogeny. To accomplish this, another isogeny of degree $2^{e'_{\mathrm{rsp}}} - q_{\mathrm{rsp}}$ originating from E_{com} must be computed via the following method. First, a

Algorithm 4.4 ComputeBacktrackingAndNormalize(α)

Input: A quaternion element α.
Output: A quaternion element α, and the backtracking n.

Write α as $\alpha = \alpha_0 + \alpha_1 i + \alpha_2 j + \alpha_3 k$
$\alpha'_0 \leftarrow \alpha_0 - \alpha_3$ // $\alpha'_0, \alpha'_1, \alpha'_2, \alpha'_3$ represents α in the $\mathcal{O}_0$-basis
$\alpha'_1 \leftarrow \alpha_1 - \alpha_2$
$\alpha'_2 \leftarrow \alpha_2/2$
$\alpha'_3 \leftarrow \alpha_3/2$
$g \leftarrow \gcd(\alpha'_0, \alpha'_1, \alpha'_2, \alpha'_3)$
$n \leftarrow$ DyadicValuation(g)
$\alpha \leftarrow \alpha/g$
return α, n

random $\mathcal{O}_0$-left ideal I_{aux} of norm $2^{e'_{\text{rsp}}} - q_{\text{rsp}}$ is sampled using RandomIdealGivenNorm. In general, the quantity $2^{e'_{\text{rsp}}} - q_{\text{rsp}}$ is not prime. This means that RandomIdealGivenNorm may fail. If this occurs, it is necessary to restart the entire signing procedure.

Let $I_{\text{com,rsp}}$ denote the ideal associated to the isogeny $\varphi^{\text{odd}}_{\text{rsp}} \circ \varphi_{\text{com}}$, i.e., $I_{\text{com,rsp}} = \mathcal{O}_0\alpha_{\text{rsp}} + \mathcal{O}_0(q_{\text{rsp}}D_{\text{mix}})$, and let $I_{\text{com,rsp,aux}} = I_{\text{com,rsp}} \cap I_{\text{aux}}$. Using IdealToIsogeny, the isogeny $\varphi_{\text{com,rsp,aux}} : E_0 \to E'_{\text{aux}}$ corresponding to $I_{\text{com,rsp,aux}}$ is generated. It is worth mentioning that IdealToIsogeny may fail. If this occurs, it is necessary to restart the entire signing procedure.

The two-dimensional isogeny Φ with kernel

$$\langle([2^{f-e'_{\text{rsp}}}]\varphi_{\text{com}}(P_0), [q_{\text{rsp}}^{-1}2^{f-e'_{\text{rsp}}}]\varphi_{\text{com,rsp,aux}}(P_0)), ([2^{f-e'_{\text{rsp}}}]\varphi_{\text{com}}(Q_0), [q_{\text{rsp}}^{-1}2^{f-e'_{\text{rsp}}}]\varphi_{\text{com,rsp,aux}}(Q_0))\rangle$$

is represented by the following matrix

$$\Phi = \begin{pmatrix} \varphi_{\text{aux}} & -\widehat{\psi} \\ \varphi^{\text{odd}}_{\text{rsp}} & \widehat{\varphi'}_{\text{aux}} \end{pmatrix} : E_{\text{com}} \times E'_{\text{aux}} \to E_{\text{aux}} \times E_{\text{chl}},$$

where the isogenies $\varphi_{\text{aux}} : E_{\text{com}} \to E_{\text{aux}}$ and $\psi : E_{\text{chl}} \to E'_{\text{aux}}$ fit into the diagram Figure 4.

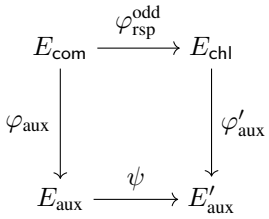

FIGURE 4. Auxiliary isogeny diagram.

By evaluating the isogeny Φ at the pair of points $([2^{f-e'_{\text{rsp}}-r_{\text{rsp}}}]\varphi_{\text{com}}(P_0), 0_{E'_{\text{aux}}})$ and $([2^{f-e'_{\text{rsp}}-r_{\text{rsp}}}]\varphi_{\text{com}}(Q_0), 0_{E'_{\text{aux}}})$, it is possible to obtain the images

- $[2^{f-e'_{\text{rsp}}-r_{\text{rsp}}}]\varphi_{\text{aux}} \circ \varphi_{\text{com}}(P_0)$,
- $[2^{f-e'_{\text{rsp}}-r_{\text{rsp}}}]\varphi_{\text{aux}} \circ \varphi_{\text{com}}(Q_0)$,
- $[2^{f-e'_{\text{rsp}}-r_{\text{rsp}}}]\varphi^{\text{odd}}_{\text{rsp}} \circ \varphi_{\text{com}}(P_0)$, and
- $[2^{f-e'_{\text{rsp}}-r_{\text{rsp}}}]\varphi^{\text{odd}}_{\text{rsp}} \circ \varphi_{\text{com}}(Q_0)$.

In practice, the two-dimensional isogeny Φ is computed using the algorithm Isogeny22ChainWithTorsion, which, as explained in Section 2.4.1, requires additional torsion above the kernel generators. The algorithm to evaluate the isogenies φ_{aux} and $\varphi^{\text{odd}}_{\text{rsp}}$ on the $2^{e'_{\text{rsp}}+r_{\text{rsp}}+2}$-torsion is summarized in SplitAuxiliaryIsogeny. From here on, we write $(P_{\text{aux}}, Q_{\text{aux}})$ to denote the points $\left([2^{f-e'_{\text{rsp}}-r_{\text{rsp}}-2}]\varphi_{\text{aux}} \circ \varphi_{\text{com}}(P_0),\ [2^{f-e'_{\text{rsp}}-r_{\text{rsp}}-2}]\varphi_{\text{aux}} \circ \varphi_{\text{com}}(Q_0)\right)$, and we write $(P_{\text{chl}}, Q_{\text{chl}})$ to denote the points $\left([2^{f-e'_{\text{rsp}}-r_{\text{rsp}}-2}]\varphi^{\text{odd}}_{\text{rsp}} \circ \varphi_{\text{com}}(P_0),\ [2^{f-e'_{\text{rsp}}-r_{\text{rsp}}-2}]\varphi^{\text{odd}}_{\text{rsp}} \circ \varphi_{\text{com}}(Q_0)\right)$.

If $e'_{\text{rsp}} = e_{\text{rsp}} - r_{\text{rsp}} - n_{\text{bt}} = 0$, it means that $\varphi_{\text{rsp}} = \varphi^{\text{even}}_{\text{rsp}}$. As a consequence, there is no need to compute any two-dimensional isogeny or auxiliary isogeny. In this case, the ideal $I_{\text{com,rsp}}$ is translated to the isogeny $\varphi_{\text{com,rsp}}$ via

Algorithm 4.5 SplitAuxiliaryIsogeny$(E_1, E_2, P_1, Q_1, P_2, Q_2, q, e', r)$

Input: The curves E_1 and E_2, the points P_1, Q_1, P_2 and Q_2, the positive integers q, e' and r such that there exist four isogenies $\varphi_q : E_1 \to F_2$, $\varphi_{2^{e'}-q} : F_2 \to E_2$, $\psi_{2^{e'}-q} : E_1 \to F_1$ and $\psi_q : F_1 \to E_2$ verifying $\deg(\varphi_d) = \deg(\psi_d) = d$, $\varphi = \varphi_{2^{e'}-q} \circ \varphi_q = \psi_q \circ \varphi_{2^{e'}-q}$ and $(P_2, Q_2) = (\varphi(P_1), \varphi(Q_1))$.
Output: The curve F_1 together with $(R_1, S_1) = ([2^{f-e'-r-2}]\psi_{2^{e'}-q}(P_1), [2^{f-e'-r-2}]\psi_{2^{e'}-q}(Q_1))$, the curve F_2 together with $(R_2, S_2) = ([2^{f-e'-r-2}]\varphi_q(P_1), [2^{f-e'-r-2}]\varphi_q(Q_1))$.

1: $P_1'', Q_1'' \leftarrow [2^{f-e'-r-2}]P_1, [2^{f-e'-r-2}]Q_1$
2: $P_1', Q_1' \leftarrow [2^r]P_1'', [2^r]Q_1''$
3: $q_{\text{inv}} \leftarrow q^{-1} \pmod{2^{f-e'-2}}$
4: $P_2', Q_2' \leftarrow [q_{\text{inv}}2^{f-e'-2}]P_2, [q_{\text{inv}}2^{f-e'-2}]Q_2$
5: $F_1 \times F_2, \{(S_1, S_2), (R_1, R_2)\} \leftarrow$ Isogeny22ChainWithTorsion$((P_1', P_2'), (Q_1', Q_2'), \{(P_1'', 0_{E_2}), (Q_1'', 0_{E_2})\})$
6: **return** $F_1, S_1, R_1, F_2, S_2, R_2$

IdealToIsogeny. As already mentioned, the algorithm IdealToIsogeny may fail; if that is the case, the signing algorithm restarts with fresh randomness. Let $(P_{\mathsf{chl}}, Q_{\mathsf{chl}})$ denote $([2^{f-e'_{\text{rsp}}-2}]\varphi_{\text{com,rsp}}(P_0), [2^{f-r_{\text{rsp}}-2}]\varphi_{\text{com,rsp}}(Q_0))$. Additionally, the values $E_{\mathsf{aux}}, P_{\mathsf{aux}}, Q_{\mathsf{aux}}$ are set up to be equal to $E_{\mathsf{chl}}, P_{\mathsf{chl}}, Q_{\mathsf{chl}}$.

As mentioned above, the isogeny $\varphi_{\text{rsp}}^{\text{even}}$ is represented via its kernel. This is only necessary if $r_{\text{rsp}} > 0$. In such an instance, the kernel $\varphi_{\text{rsp}}^{\text{even}}$ is computed as follows. It is worth noting that if the kernel of the isogeny corresponding to the ideal $\mathcal{O}_0\alpha_{\text{rsp}} + \mathcal{O}_0(2^{r_{\text{rsp}}})$ coincides with $\langle [2^{f-r_{\text{rsp}}}s]P_0 + [2^{f-r_{\text{rsp}}}t]Q_0\rangle$ for some integers s and t, then $\ker(\varphi_{\text{rsp}}^{\text{even}}) = \langle [2^{e'_{\text{rsp}}+2}s]P_{\mathsf{chl}} + [2^{e'_{\text{rsp}}+2}t]Q_{\mathsf{chl}}\rangle$. The values s and t can be computed by running IdealToKernel on input $\mathcal{O}_0\alpha_{\text{rsp}} + \mathcal{O}_0(2^{r_{\text{rsp}}})$.

Once the values s and t have been computed, the isogeny $\varphi : E_{\mathsf{chl}} \to E$ with kernel $\langle [2^{e'_{\text{rsp}}+2}s]P_{\mathsf{chl}} + [2^{e'_{\text{rsp}}+2}t]Q_{\mathsf{chl}}\rangle$ is generated. Then, the curve E_{chl} is updated to E, and the points P_{chl} and Q_{chl} are updated to $\varphi(P_{\mathsf{chl}})$ and $\varphi(Q_{\mathsf{chl}})$, respectively. This step is summarized in ComputeEvenNonBacktrackingResponse.

Algorithm 4.6 ComputeEvenNonBacktrackingResponse(E, P, Q, α, e', r)

Input: The curve E, the points P and Q, the quaternion α and the integers e' and r.
Output: The curve E', the points P' and Q'.

1: $I \leftarrow \mathcal{O}_0\alpha + \mathcal{O}_0(2^r)$
2: $(s, t) \leftarrow$ IdealToKernel(I)
3: $K \leftarrow [2^{e'+2}s]P + [2^{e'+2}s]Q$
4: $E', \{P', Q'\} \leftarrow$ TwoIsogenyChainSmall$(K, E, r, \{P, Q\}, \texttt{true})$
5: **return** E', P', Q'

As mentioned above, the integer chl defines the challenge isogeny φ_{chl} originating from E_{pk}. Differently from the isogeny $\varphi_{I_{\mathsf{chl}}}$ introduced at the beginning of this section, the isogeny φ_{chl} does not include the backtracking steps. To be more precise, the quantity chl describes the kernel $\langle [2^{n_{\text{bt}}}](P_{\mathsf{pk}} + [\mathsf{chl}]Q_{\mathsf{pk}})\rangle$; let $\varphi_{\mathsf{chl}} : E_{\mathsf{pk}} \to E'_{\mathsf{chl}}$ be the isogeny whose kernel equals $\langle [2^{n_{\text{bt}}}](P_{\mathsf{pk}} + [\mathsf{chl}]Q_{\mathsf{pk}})\rangle$. Even though the curves E_{chl} and E'_{chl} are isomorphic, they may be distinct. The points P_{chl} and Q_{chl} are mapped onto E'_{chl} via IsomorphismMontgomeryCurves and updated accordingly. The computation of the challenge isogeny is summarized in ComputeChallengeIsogeny.

Algorithm 4.7 ComputeChallengeIsogeny$(E, ch, P, Q, E', P', Q', n)$

Input: The curve E, an integer ch describing the kernel of an isogeny originating from E, the points P, Q on E, the curve E' together with two points P' and Q' on E', and the integer n.
Output: A curve E'' together with two points P'' and Q''.

1: $E'', \_ \leftarrow$ TwoIsogenyChain$([2^n](P + [ch]Q), E, f - n, _)$
2: $P'', Q'' \leftarrow$ IsomorphismMontgomeryCurves(E', P', Q', E'')
3: **return** E'', P'', Q''

The final part of the response generation is dedicated to obtaining a suitable representation of the data included in the signature. If $e'_{\text{rsp}} > 0$, the points P_{aux}, Q_{aux}, P_{chl} and Q_{chl} describe the kernel of a two-dimensional isogeny embedding the odd part of the response isogeny, namely $\varphi_{\text{rsp}}^{\text{odd}}$. The function SetChangeOfBasisMatrix is

devoted to encoding these points in a matrix. The first step of this process consists of generating two deterministic bases $(P_{\mathsf{aux}}^{\text{det}}, Q_{\mathsf{aux}}^{\text{det}})$ and $(P_{\mathsf{chl}}^{\text{det}}, Q_{\mathsf{chl}}^{\text{det}})$ of E_{aux} and E_{chl}, respectively, via TorsionBasisToHint; the hints $\text{hint}_{\mathsf{aux}}$ and $\text{hint}_{\mathsf{chl}}$ output by TorsionBasisToHint are also part of the signature. Then, the change-of-basis matrix $\mathbf{M}_{\mathsf{aux}}$ between $(P_{\mathsf{aux}}, Q_{\mathsf{aux}})$ and $([2^{f-e'_{\text{rsp}}-r_{\text{rsp}}-2}]P_{\mathsf{aux}}^{\text{det}}, [2^{f-e'_{\text{rsp}}-r_{\text{rsp}}-2}]Q_{\mathsf{aux}}^{\text{det}})$ is computed using ChangeOfBasis. The basis $(P_{\mathsf{chl}}^{\text{det}}, Q_{\mathsf{chl}}^{\text{det}})$ is multiplied by $\mathbf{M}_{\mathsf{aux}}$ and then scaled by $2^{f-e'_{\text{rsp}}-r_{\text{rsp}}-2}$. The result will serve as input for the algorithm ChangeOfBasis that outputs the change-of-basis-matrix $\mathbf{M}_{\mathsf{chl}}$.

Algorithm 4.8 SetChangeOfBasisMatrix$(E_1, E_2, P_1, Q_1, P_2, Q_2, e)$

Input: The elliptic curves E_1 and E_2, the points P_1 and Q_1 on E_1, the points P_2 and Q_2 on E_2, and an integer e.
Output: A change-of-basis $\mathbf{M}_2$ and the hints hint_1 and hint_2.

1: $P'_1, Q'_1, \text{hint}_1 \leftarrow$ TorsionBasisToHint(E_1)
2: $P'_2, Q'_2, \text{hint}_2 \leftarrow$ TorsionBasisToHint(E_2)
3: $P'_1, Q'_1 \leftarrow [2^{f-e-2}]P'_1, [2^{f-e-2}]Q'_1$
4: $P'_2, Q'_2 \leftarrow [2^{f-e-2}]P'_2, [2^{f-e-2}]Q'_2$
5: $\mathbf{M}_1 \leftarrow$ ChangeOfBasis$_{2^e}(E_1, (P_1, Q_1), (P'_1, Q'_1))$
6: $(P_2, Q_2) \leftarrow \mathbf{M}_1 \cdot (P_2, Q_2)$
7: $\mathbf{M}_2 \leftarrow$ ChangeOfBasis$_{2^e}(E_2, (P'_2, Q'_2), (P_2, Q_2))$
8: **return** $\mathbf{M}_2, \text{hint}_1, \text{hint}_2$

Finally, the signature σ consists in the following data.

(1) The auxiliary curve E_{aux};
(2) the integer n_{bt}, describing the number of backtracking steps;
(3) the integer r_{rsp}, describing the largest integer such that $2^{n_{\text{bt}}}$ divides the degree of the response isogeny;
(4) the change-of-basis matrix $\mathbf{M}_{\mathsf{chl}}$;
(5) the challenge chl;
(6) the hint $\text{hint}_{\mathsf{aux}}$ on the auxiliary curve;
(7) the hint $\text{hint}_{\mathsf{chl}}$ on the challenge curve.

4.5. Verification

The algorithm SQIsign.Verify (Algorithm 4.9) describes the verification procedure. It checks whether the signature is valid by evaluating the response isogeny $\varphi_{\text{rsp}} : E_{\mathsf{com}} \to E_{\mathsf{chl}}$. On input a message msg, a signature $\sigma = (E_{\text{aux}}, n_{\text{bt}}, r_{\text{rsp}}, \mathbf{M}_{\mathsf{chl}}, \mathsf{chl}, \text{hint}_{\mathsf{aux}}, \text{hint}_{\mathsf{chl}})$, and a public key $\mathsf{pk} = (E_{\mathsf{pk}}, \text{hint}_{\mathsf{pk}})$, the verifier first checks that the curves E_{pk} and E_{aux} are indeed supersingular.

In this setting, to check that a curve E is supersingular, it suffices to verify that there exists a basis of $E[2^f]$. This translates to proving that bases generated from the hints $\text{hint}_{\mathsf{pk}}$, $\text{hint}_{\mathsf{aux}}$ and $\text{hint}_{\mathsf{chl}}$ span the entire 2^f-torsion. The points computed for these bases serve as kernel generators for one- and two-dimensional isogenies. Therefore, when computing such isogenies, the verifier additionally checks that the input is a correct basis. For efficiency reasons, these checks are not explicit in the implementation, but rather a byproduct of a successful computation of an isogeny. We refer to Section 8.4 and Section 8.5 for more details.

The formulas to compute the two-dimensional isogenies discussed in Section 2.4 require that kernels are isotropic. Hence, to ensure that the algorithms employed during verification behave correctly, it is fundamental to additionally check this condition; we refer to Section 8.5.3 for further detail on this step.

Before computing a representation for the response isogeny, the verifier needs to check that the data received is valid. In particular, the verifier checks that the quantity $e'_{\text{rsp}} = e_{\text{rsp}} - n_{\text{bt}} - r_{\text{rsp}} \geq 0$ and that all the components of $\mathbf{M}_{\mathsf{chl}}$ are positive integer $< 2^{e'_{\text{rsp}}+r_{\text{rsp}}+2}$.

After all the above checks, the verifier computes the deterministic basis $(P_{\mathsf{pk}}, Q_{\mathsf{pk}})$ on E_{pk} corresponding to the hint $\text{hint}_{\mathsf{pk}}$. Such a basis is then used to generate the kernel of the challenge isogeny $\varphi_{\mathsf{chl}} : E_{\mathsf{pk}} \to E_{\mathsf{chl}}$: given the challenge data chl, the kernel $\ker(\varphi_{\mathsf{chl}})$ is equal to $\langle [2^{n_{\text{bt}}}](P_{\mathsf{pk}} + [\mathsf{chl}]Q_{\mathsf{pk}})\rangle$.

Using the hint $\text{hint}_{\mathsf{aux}}$, the verifier computes the deterministic basis $(P_{\mathsf{aux}}, Q_{\mathsf{aux}})$. This basis is then scaled down to the correct order: $(P_{\mathsf{aux}}, Q_{\mathsf{aux}}) = ([2^{f-e'_{\text{rsp}}+2}]P_{\mathsf{aux}}, [2^{f-e'_{\text{rsp}}+2}]Q_{\mathsf{aux}})$. Similarly, using the hint $\text{hint}_{\mathsf{chl}}$, the basis $(P_{\mathsf{chl}}, Q_{\mathsf{chl}})$ is computed and then scaled down: $(P_{\mathsf{chl}}, Q_{\mathsf{chl}}) = ([2^{f-e'_{\text{rsp}}-r_{\text{rsp}}-2}]P_{\mathsf{chl}}, [2^{f-e'_{\text{rsp}}-r_{\text{rsp}}-2}]Q_{\mathsf{chl}})$. To reconstruct the correct evaluation under the response isogeny, it is necessary to scale $(P_{\mathsf{chl}}, Q_{\mathsf{chl}})$ by the matrix $\mathbf{M}_{\mathsf{chl}}$, i.e., $(P_{\mathsf{chl}}, Q_{\mathsf{chl}}) = \mathbf{M}_{\mathsf{chl}} \cdot (P_{\mathsf{chl}}, Q_{\mathsf{chl}})$.

Algorithm 4.9 SQIsign.Verify($\mathsf{msg}, \sigma, \mathsf{pk}$)

Input: A message msg, signature σ and public key pk.
Output: verified a boolean indicating whether the verification passed.

Parse pk as $(E_{\mathsf{pk}}, \mathsf{hint}_{\mathsf{pk}})$
Parse σ as $(E_{\mathsf{aux}}, n_{\mathrm{bt}}, r_{\mathrm{rsp}}, \mathbf{M}_{\mathsf{chl}}, \mathsf{chl}, \mathsf{hint}_{\mathsf{aux}}, \mathsf{hint}_{\mathsf{chl}})$
Check that E_{pk} is supersingular, otherwise **return** `false`
Check that E_{aux} is supersingular, otherwise **return** `false`
Check that all the entries of $\mathbf{M}_{\mathsf{chl}}$ are positive integers $< 2^{e'_{\mathrm{rsp}}+r_{\mathrm{rsp}}+2}$, otherwise **return** `false`
$e'_{\mathrm{rsp}} \leftarrow e_{\mathrm{rsp}} - n_{\mathrm{bt}} - r_{\mathrm{rsp}}$
Check that $e'_{\mathrm{rsp}} \geq 0$, otherwise **return** `false`
$P_{\mathsf{pk}}, Q_{\mathsf{pk}} \leftarrow \mathsf{TorsionBasisFromHint}(E_{\mathsf{pk}}, \mathsf{hint}_{\mathsf{pk}})$
$E_{\mathsf{chl}}, \_ \leftarrow \mathsf{TwoIsogenyChain}([2^{n_{\mathrm{bt}}}](P_{\mathsf{pk}} + [\mathsf{chl}]Q_{\mathsf{pk}}), E_{\mathsf{pk}}, f - n_{\mathrm{bt}}, \_)$
$P_{\mathsf{aux}}, Q_{\mathsf{aux}} \leftarrow \mathsf{TorsionBasisFromHint}(E_{\mathsf{aux}}, \mathsf{hint}_{\mathsf{aux}})$
$P_{\mathsf{aux}}, Q_{\mathsf{aux}} \leftarrow [2^{f-e'_{\mathrm{rsp}}+2}]P_{\mathsf{aux}}, [2^{f-e'_{\mathrm{rsp}}+2}]Q_{\mathsf{aux}}$
$P_{\mathsf{chl}}, Q_{\mathsf{chl}} \leftarrow \mathsf{TorsionBasisFromHint}(E_{\mathsf{chl}}, \mathsf{hint}_{\mathsf{chl}})$
$P_{\mathsf{chl}}, Q_{\mathsf{chl}} \leftarrow [2^{f-e'_{\mathrm{rsp}}-r_{\mathrm{rsp}}-2}]P_{\mathsf{chl}}, [2^{f-e'_{\mathrm{rsp}}-r_{\mathrm{rsp}}-2}]Q_{\mathsf{chl}}$
$P_{\mathsf{chl}}, Q_{\mathsf{chl}} \leftarrow \mathbf{M}_{\mathsf{chl}} \cdot (P_{\mathsf{chl}}, Q_{\mathsf{chl}})$
if $r_{\mathrm{rsp}} > 0$ **then**
  if $\mathbf{M}_{\mathsf{chl}}[0][0]$ is even **and** $\mathbf{M}_{\mathsf{chl}}[0][1]$ is even **then**
    $K_{\mathsf{resp}} \leftarrow [2^{e'_{\mathrm{rsp}}+2}]Q_{\mathsf{chl}}$
  else
    $K_{\mathsf{resp}} \leftarrow [2^{e'_{\mathrm{rsp}}+2}]P_{\mathsf{chl}}$
  $E_{\mathsf{chl}}, \{P_{\mathsf{chl}}, Q_{\mathsf{chl}}\} \leftarrow \mathsf{TwoIsogenyChainSmall}(K_{\mathsf{resp}}, E_{\mathsf{chl}}, r_{\mathrm{rsp}}, \{P_{\mathsf{chl}}, Q_{\mathsf{chl}}\}, \mathtt{false})$
if $e'_{\mathrm{rsp}} = 0$ **then**
  $\mathsf{chl}' \leftarrow \mathsf{HASH}(\mathsf{pk} \mid\mid j(E_{\mathsf{chl}}) \mid\mid \mathsf{msg})$
  return $(\mathsf{chl} = \mathsf{chl}')$
Check that $(([4]P_{\mathsf{chl}}, [4]P_{\mathsf{aux}}), ([4]Q_{\mathsf{chl}}, [4]Q_{\mathsf{aux}}))$ is isotropic, otherwise **return** `false` // See Section 8.5.3
try
    $F_1 \times F_2, \_ \leftarrow \mathsf{Isogeny22ChainWithTorsion}((P_{\mathsf{chl}}, P_{\mathsf{aux}}), (Q_{\mathsf{chl}}, Q_{\mathsf{aux}}), \_)$
except
  return `false`
$\mathsf{chl}' \leftarrow \mathsf{HASH}(\mathsf{pk} \mid\mid j(F_1) \mid\mid \mathsf{msg})$
return $(\mathsf{chl} = \mathsf{chl}')$

If $r_{\mathrm{rsp}} > 0$, this means that the response isogeny computed in the signing process had an even component. It is then necessary to compute the dual of the even component. Its kernel corresponds to $\langle [2^{e'_{\mathrm{rsp}}+2}]P_{\mathsf{chl}}, [2^{e'_{\mathrm{rsp}}+2}]Q_{\mathsf{chl}} \rangle$; let $\varphi : E_{\mathsf{chl}} \to E'_{\mathsf{chl}}$ be the isogeny with this kernel and redefine $(P_{\mathsf{chl}}, Q_{\mathsf{chl}}) = (\varphi(P_{\mathsf{chl}}), \varphi(Q_{\mathsf{chl}}))$. The curve E_{chl} is updated to E'_{chl}.

In the case that $e'_{\mathrm{rsp}} = e_{\mathrm{rsp}} - n_{\mathrm{bt}} - r_{\mathrm{rsp}} = 0$ (this happens with negligible probability, but it is technically possible), there is no need to compute a two-dimensional isogeny. In this case, the curve E_{chl} coincides with the commitment curve computed by the prover.

The final step then consists in checking that $\mathsf{HASH}(\mathsf{pk} \mid\mid j(E_{\mathsf{chl}}) \mid\mid \mathsf{msg})$ is equal to chl.

Now, assume that $e'_{\mathrm{rsp}} = e_{\mathrm{rsp}} - n_{\mathrm{bt}} - r_{\mathrm{rsp}} > 0$. If the signature has been honestly generated, the couples of points $(P_{\mathsf{chl}}, P_{\mathsf{aux}})$ and $(Q_{\mathsf{chl}}, Q_{\mathsf{aux}})$ denote the 4-torsion above the kernel of a two-dimensional isogeny. Let Φ be such an isogeny and let $F_1 \times F_2$ its codomain. The curve F_1 is isomorphic to the commitment curve generated during the signing procedure, if the signature is valid. The last step of the verification process consists in recomputing the challenge chl' as $\mathsf{HASH}(\mathsf{pk} \mid\mid j(E_{\mathsf{chl}}) \mid\mid \mathsf{msg})$. If $\mathsf{chl}' = \mathsf{chl}$, the signature is valid, and it corresponds to an honest execution of the signing algorithm.

4.6. 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.
- Elliptic curves are encoded by their Montgomery coefficient $A \in \mathbb{F}_{p^2}$.
- Ideals I are encoded concatenating the encoding of $\mathrm{nrd}(I)$, which is a `FP_ENCODED_BYTES`-byte integer, with the encoding of the quaternion output by IdealGenerator(I). This quaternion is encoded by the concatenation of four `FP_ENCODED_BYTES`-byte integers, denoting the integer coefficients in the $1, i, j, k$-basis.[1]
- 2×2 integer matrices $\left(\begin{smallmatrix} a & b \\ c & d \end{smallmatrix}\right)$ are encoded by the concatenation of the encoding of a, b, c and d.
- Hints consist of two values (h_A, h); the quantity h_A is either zero or one, whereas h is a 7-bit integer. A hint is then encoded in one byte, where the least significant bit represents the flag h_A.

Public keys. The public key is encoded by concatenating the encodings of E_{pk} from Step 5 of SQIsign.KeyGen (Algorithm 4.1) together with the one of the hint $\mathsf{hint}_{\mathsf{pk}}$ from Step 8 of SQIsign.KeyGen (Algorithm 4.1).

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

- the encoding of the public key pk;
- the ideal I_{sk} from Step 4 of SQIsign.KeyGen (Algorithm 4.1);
- the 2×2 integer matrix $\mathbf{M}_{\mathsf{sk}}$ from Step 9 of SQIsign.KeyGen (Algorithm 4.1); the number of bytes to represent each matrix component is `TORSION_2POWER_BYTES`, which is a quantity specified in Appendix B for each security level.

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

- the Montgomery coefficient of E_{aux} from either Step 27 or Step 31 of SQIsign.Sign (Algorithm 4.2);
- the one-byte integer n_{bt} from Step 15 of SQIsign.Sign (Algorithm 4.2);
- the one-byte integer r_{rsp} from Step 19 of SQIsign.Sign (Algorithm 4.2);
- the 2×2 integer matrix $\mathbf{M}_{\mathsf{chl}}$ from Step 37 of SQIsign.Sign (Algorithm 4.2); each component is represented by $\lfloor (e_{\mathrm{rsp}} + 7)/8 \rfloor$ bytes.
- the $\lambda/8$-byte integer chl from Step 10 of SQIsign.Sign (Algorithm 4.2);
- the hint $\mathsf{hint}_{\mathsf{aux}}$ from Step 37 of SQIsign.Sign (Algorithm 4.2);
- the hint $\mathsf{hint}_{\mathsf{chl}}$ from Step 37 of SQIsign.Sign (Algorithm 4.2).

[1] The encoding of the quaternion does not include any denominator. The quaternion γ output by IdealGenerator(I) has a denominator r that is invertible modulo the norm $N := \mathrm{nrd}(I)$. This implies that the pair $(N, r \cdot \gamma)$ generates I as an ideal, and thus the coefficients can be scaled to be integral.

CHAPTER 5

Parameter sets

This chapter describes parameter requirements and lists the parameter sets for the instantiations of SQIsign in the NIST-I, NIST-III, and NIST-V security levels.

5.1. Parameter requirements

Each parameter set consists of a prime p (the characteristic of the finite field), and a hash function HASH (a collision-resistant hash function used to apply the Fiat-Shamir transform). All other parameters can be deduced from p and the above specification. The prime p is selected under the following considerations:

(1) Given a security parameter λ, we need prime sizes of $\log_2(p) \approx 2\lambda$ bits. Then, the best known attacks have a complexity at least $O(\sqrt{p}) = O(2^\lambda)$ (see Chapter 10).
(2) To maximize efficiency, we aim for $2p$ to fit into a number of 64 bits words no greater than $2\lambda/64$. Hence, we aim for primes p of at most 255, 383, resp. 511 bits for NIST security levels I, III, resp. V.
(3) We restrict to primes $p \equiv 3 \pmod 4$, which significantly simplifies implementations and is beneficial for fast field arithmetic over $\mathbb{F}_{p^2}$.
(4) The prime should be of the form $p = c \cdot 2^f - 1$ with c as small as possible; it is also desirable that c has small Hamming weight.
(5) The hash function HASH is used to generate the challenge. The length of the output of the hash function is the length of the challenge e_{chl}. This quantify is computed as $e_{\text{chl}} = f - \lceil \log_2 (\sqrt{p}) \rceil$ (See Section 4.2.1). Since this quantity is typically slightly smaller than the targeted security parameter λ, we apply the *grinding* technique: to compensate for the $\lambda - e_{\text{chl}}$ missing bits, the hash function is constructed by iterating $2^{\lambda - e_{\text{chl}}}$ times a standard cryptographic hash function.

5.2. Parameter sets

In this section, we list parameter sets for the NIST-I, NIST-III, and NIST-V security levels. A parameter set consists of a targeted security parameter λ, a prime p and a hash function HASH. All other parameters can be deduced from p and the above specification; for convenience, we also list below the values of f and c such that $p = c \cdot 2^f - 1$. All other derived values are listed in Appendix B.

Let n be any positive integer. In the following, we write $\mathsf{SHAKE256}_n$ for the first n bits of $\mathsf{SHAKE256}$. For any function $f : A \to B$, with $B \subseteq A$, we write $f^{\circ n} = f \circ \cdots \circ f$ for f iterated n times.

NIST-I:

$$\begin{aligned} \lambda^{\text{I}} &= 128 \\ p^{\text{I}}_{5248} &= 5 \cdot 2^{248} - 1 \\ &= \texttt{0x4fffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff} \\ \mathsf{HASH} &= \mathsf{SHAKE256}_{122} \circ \mathsf{SHAKE256}^{\circ 63}_{256} \end{aligned}$$

NIST-III:

$$\lambda^{\mathrm{III}} = 192$$

$$p^{\mathrm{III}}_{65376} = 65 \cdot 2^{376} - 1$$

$$= \texttt{0x40ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff}$$
$$\texttt{ffffffffffffffffffffffffffffffffff}$$

$$\mathsf{HASH} = \mathsf{SHAKE256}_{184} \circ \mathsf{SHAKE256}^{\circ 255}_{384}$$

NIST-V:

$$\lambda^{\mathrm{V}} = 256$$

$$p^{\mathrm{V}}_{27500} = 27 \cdot 2^{500} - 1$$

$$= \texttt{0x1afffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff}$$
$$\texttt{ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff}$$

$$\mathsf{HASH} = \mathsf{SHAKE256}_{247} \circ \mathsf{SHAKE256}^{\circ 511}_{512}$$

CHAPTER 6

Known answer test values

The folder **KAT** of the submission media file contains KAT files for SQIsign parameters corresponding to security levels 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 7

Performance analysis

The submission package includes a reference implementation written in portable C (C11) and an additional, assembly-optimized implementation targeting the Intel Broadwell architecture (and later). 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:

- `signature`: implementing the signature key-generation and signing protocols.
- `verification`: implementing the signature verification protocol.
- `id2iso`: implementation of ideal to isogeny algorithms.
- `ec`: module for the elliptic curve and isogeny computations.
- `quaternion`: module for quaternion computation. This module employs arbitrary precision arithmetic based on GMP, and floating-point arithmetic using DPE.
- `precomp`: module to pre-compute constants for the code package.
- `gf`: implementation of the finite field arithmetic over $\mathbb{F}_p$ and $\mathbb{F}_{p^2}$.
- `hd`: computations with (2,2)-isogenies in the theta model.
- `mp`: routines for arbitrary precision arithmetic of integers with no dependency of GMP.
- `common`: common dependencies and symmetric primitive implementations.

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 big integers and floating point arithmetic in the Quaternion module), it is not constant-time.

7.1. Key and signature sizes

Key and signature sizes are listed in Table 4 for each security level.

Table 4. SQIsign key and signature sizes in bytes for each security level.

Parameter set	Public key	Secret key	Signature
NIST-I	65	353	148
NIST-III	97	529	224
NIST-V	129	701	292

7.2. 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 -DGMP_LIBRARY:STRING=BUILD -DGMP_BUILD_CONFIG_ARGS:STRING="-disable-assembly"`

7.3. Optimized implementation

The optimized implementation is the same as the reference implementation.

7.4. Intel Broadwell optimized implementation

The additional implementation uses assembly-optimized code targeting the Intel Broadwell architecture (and later). The assembly optimizations are applied to the GF-module, while the remaining code base is re-used from the reference implementation; see Section 8.1 for more details. For benchmarking, we used the following CMake options: `-DSQISIGN_BUILD_TYPE=broadwell -DCMAKE_BUILD_TYPE=Release`.

7.5. ARM Cortex-M4 implementation

The verification routines of the reference implementation are compatible with 32-bit embedded systems based on e.g., the ARM Cortex-M4 core. However, the build system and provided applications are not suitable for a bare-metal environment as typically provided in such systems. The recommended platform for evaluating SQIsign on the ARM Cortex-M4 core is the `pqm4` project [KPR+]. A helper shell script, to copy the source files to a folder structure compatible with `pqm4`, is provided in the submission package; instructions for its use can be found in the `README.md` file.

It is noted that `pqm4` assumes that all API functions (key pair generation, signature and verification) are available, but this submission only provides verification routines for embedded systems. To work around this issue, key pair generation and signature routines are mocked to return a fixed key pair and signatures of fixed messages using that key pair, allowing a verification-only implementation to be evaluated. A GitHub repository is provided, containing a snapshot of `pqm4` at the time of the submission, incorporating the verification routines for SQIsign for NIST-I, III and V parameter sets; these can be found in the `sqisign` branch of that repository, which is available at the following URL:

- `https://github.com/SQISign/the-sqisign-pqm4`.

7.6. Performance evaluation

All builds below are compiled with clang 18, using compile flags `-march=native -O3` and GMP version 6.3.0, unless otherwise noted.

Performance evaluation of both the reference and assembly-optimized implementations was performed on an Intel Core i7-13700K (13th generation, codename Raptor Lake) x86-64 CPU with nominal core clock of 3.4 GHz, running Ubuntu Linux 24.04 LTS. Turbo Boost and HyperThreading were turned off during benchmarking to get consistent timings. Comparing the current system to the machine specified in Round 1, which used an Intel Xeon Gold 6338 (Ice Lake), shows a performance improvement of approximately 15% in terms of clock cycles. The results are shown in Table 5.

Performance of the reference implementation was also evaluated on 64-bit ARM platforms, specifically, on an Apple M3 Max CPU (performance core) with nominal core clock of 4.05 GHz, running macOS 15.2. The compiler used is Apple clang 16. The results are shown in Table 6.

Lastly, performance, stack usage and code size of the verification routines were evaluated for the reference implementation, using the `pqm4` project, on an ST Microelectronics NUCLEO-L4R5ZI evaluation board for the 32-bit ARM Cortex-M4-based STM32L4R5ZI microcontroller. The results are shown in Table 7. As all other schemes included in pqm4, to our knowledge, include all NIST API routines (key pair generation, signing and verification), care should be taken when comparing stack usage and code size figures to this verification-only implementation of SQIsign.

TABLE 5. SQISIGN performance in 10^6 CPU cycles on an Intel Core i7-13700K CPU. Results are the median of 1,000 benchmark runs.

Parameter set	`KeyGen`	`Sign`	`Verify`
Reference implementation (with default GMP installation)			
NIST-I	71.8	163.1	11.3
NIST-III	188.2	427.0	30.4
NIST-V	325.4	751.8	61.9
Reference implementation (with GMP `--disable-assembly`)			
NIST-I	84.4	203.1	11.3
NIST-III	227.9	548.9	30.5
NIST-V	402.6	1021.0	62.2
Assembly-optimized implementation for Intel Broadwell or later			
NIST-I	43.3	101.6	5.1
NIST-III	134.0	309.2	18.6
NIST-V	212.0	507.5	35.7

TABLE 6. SQISIGN performance in 10^6 CPU cycles on an Apple M3 Max CPU. Results are the median of 1,000 benchmark runs.

Parameter set	`KeyGen`	`Sign`	`Verify`
Reference implementation (with default GMP installation)			
NIST-I	48.8	114.8	8.0
NIST-III	138.7	316.4	22.6
NIST-V	241.1	556.3	46.8
Reference implementation (with GMP `--disable-assembly`)			
NIST-I	57.5	138.6	8.0
NIST-III	163.6	396.4	22.6
NIST-V	293.4	732.6	46.8

TABLE 7. SQISIGN verification performance, stack usage and code size on an ARM Cortex-M4-based microcontroller, benchmarked using the `pqm4` project, compiled with gcc version 13.2 with `pqm4`'s preset optimization settings for speed. Results are obtained from a single run of the benchmark.

Parameter set	`Verify`		
	10^6 CPU cycles	Stack usage (KB)	Code size (KB)
NIST-I	123	31	40
NIST-III	375	50	44
NIST-V	751	64	46

CHAPTER 8

Implementation details

This chapter describes implementation details of the underlying algorithms that are required to construct SQIsign. This includes implementation details of the field, elliptic-curve, pairing and isogeny arithmetic.

We note that, although the implementations provided in this submission do not run in constant-time, we have made an effort to select and implement most of the algorithms covered in this chapter in constant-time.

The implementations included in the submission package support all field arithmetic operations described in Section 2.1. In addition to the reference implementation, there are two platform-specific implementations, one for 64-bit Intel processors, and another for 64-bit ARM processors supporting that architecture's Cryptographic Extensions. The only changes in these platform-specific implementations, compared to the reference implementations, are in the field arithmetic layer and the pseudo-random number generation. Concretely:

- In the reference implementation, operations over $\mathbb{F}_p$ and $\mathbb{F}_{p^2}$ are written in plain C, and pseudo-random number generation uses operating system-specific facilities, such as reading from '/dev/[u]random' in Unix-based systems.
- In the 64-bit Intel implementation, addition, multiplication and squaring operations over $\mathbb{F}_p$ and $\mathbb{F}_{p^2}$ are written in assembly language using optimized algorithms [Lon23]; these are described in Section 8.1. Pseudo-random number generation uses the AES256-CTR-DRBG algorithm, accelerated using AES-NI instructions.
- In the 64-bit ARM implementation, the same plain C field arithmetic code as in the reference implementation is used, and pseudo-random number generation uses the AES256-CTR-DRBG algorithm, accelerated using ARM Cryptographic Extensions instructions.

Speedups of up to 10% for keypair generation and signing were observed through the use of accelerated AES256-CTR-DRBG in the 64-bit Intel and 64-bit ARM implementations.

8.1. Finite field arithmetic

Since isogeny computations dominate the execution time of signature and verification, and rely heavily on arithmetic in both $\mathbb{F}_{p^2}$ and its underlying field $\mathbb{F}_p$, faster arithmetic in both fields directly leads to faster SQIsign performance.

Both the reference and 64-bit Intel implementations utilize Montgomery arithmetic. The reference implementation uses an *unsaturated* or *reduced-radix* representation, whereas the 64-bit Intel implementation uses a *saturated* or *full-radix* representation. Both implementations exploit the fact that SQIsign primes have the special shape $p = \beta \cdot 2^\alpha - 1$, with a relatively small integer β and a large integer α. Primes with this characteristic are often referred to as *Montgomery-friendly primes*, which allow efficient Montgomery reduction; we refer to [FHLO-JRH18; Lon23] for further details on this technique. The exceptionally high degree of Montgomery-friendliness of the three SQIsign primes enabled us to develop particularly compact and performant implementations.

8.1.1. Reference implementation

We used Scott's automatic code generator [Sco24] to synthesize portable C code for $\mathbb{F}_p$ arithmetic within the SQIsign reference library. This generator employs an *unsaturated* representation, using radix 2^r with $r < 32$ (for 32-bit CPUs) or $r < 64$ (for 64-bit CPUs). We utilized this generator to produce code for all three primes specified in Section 5.2 for the NIST-I, NIST-III, and NIST-V security levels.

As stated above, Montgomery reduction is sped up by exploiting the Montgomery-friendly shape of the SQIsign primes. A second source of speedup for Scott's code generator is the avoidance of the final conditional subtraction in Montgomery reduction, which is possible if the Montgomery parameter R is chosen so that $R > 4p$ [Sco24].

8.1.2. 64-bit Intel implementation

Our 64-bit Intel implementation performs $\mathbb{F}_{p^2}$ multiplication using a generalization of the interleaved radix-r Montgomery multiplication technique proposed in [Lon23]. Specifically, we implement [Lon23, Algorithm 5] after specializing it to SQIsign primes and setting $B = 1$; the resulting algorithm OptimizedPartialFp2Mul is presented as Algorithm 8.1. This algorithm employs the coarsely integrated form to interleave the multiplication and reduction parts.

Given a multiplication in $\mathbb{F}_{p^2}$ with the form $c = a \cdot b = c_0 + c_1 \cdot i$, where $c_0 = a_0b_0 - a_1b_1$ and $c_1 = a_0b_1 + a_1b_0$, we compute separately the terms c_0 and c_1 using Algorithm 8.1. For the computation of c_1, we use (a_0, b_1) and (a_1, b_0) as input pairs to Algorithm 8.1. Similarly, for the computation of c_0, we use (a_0, b_0) and $(a_1, p - b_1)$ (or $(a_1, 2p - b_1)$, resp.) as input pairs to Algorithm 8.1 for the NIST-I (NIST-III and NIST-V, resp.) security level; note that we use negative values of b_1 to guarantee a positive result. We pick either p or $2p$ for efficiency purposes and based on a bound analysis. Based on this analysis, we are also able to eliminate or simplify the final conditional subtraction, i.e., steps 6-7 of Algorithm 8.1, for NIST-III and NIST-V. We use radix $r = 2^w = 2^{64}$.

For the implementation, we exploit the MULX and ADX instructions, which are available in Intel 5th generation core (codename Broadwell) processors and later. The specialized carry manipulation allowed by these instructions enables an efficient implementation of multiply-add chains, as found in line 3 of Algorithm 8.1.

Algorithm 8.1 OptimizedPartialFp2Mul$((a_0, a_1), (b_0, b_1))$

Input: Integer input pairs (a_0, a_1) and (b_0, b_1) s.t. $a_i, b_i \in [0, p)$ and $0 \leq (a_0b_0 + a_1b_1) < pR$, where $R = 2^{nw}$, $p = \beta \cdot 2^\alpha - 1$, $n = \lceil l/w \rceil$, $l = \lceil \log_2 p \rceil$, and w is the computer wordsize; $z = \lceil \alpha/w \rceil$, $\hat{p} = (p+1)/2^{zw}$, and the radix $r = 2^w$. Integers are represented in radix r, e.g., $a_i = (a_{i,n-1}, \ldots, a_{i,1}, a_{i,0})_r$.

Output: The Montgomery residue $c = (a_0b_0 + a_1b_1) \cdot R^{-1} \bmod p$ s.t. $c \in [0, p)$

1: $u \leftarrow 0$
2: **for** j **from** 0 **up to** $n - 1$ **do**
3: $\quad u \leftarrow u + a_{0,j} \cdot b_0 + a_{1,j} \cdot b_1$
4: $\quad q \leftarrow u \bmod 2^w$
5: $\quad u \leftarrow \lfloor u/2^w \rfloor + 2^{(z-1)w} q \cdot \hat{p}$
6: **if** $u \geq p$ **then**
7: $\quad u \leftarrow u - p$
8: **return** $c \leftarrow u$

8.1.3. Other optimized field operations

For the remaining operations in $\mathbb{F}_p$, we adopt different strategies for the reference and optimized versions. The reference code implements exponentiation with exponent $(p - 3)/4$ in constant-time through an addition chain for primes of form $p \equiv 3 \bmod 4$. The result is then reused to compute inversions, Legendre symbols and square roots in $\mathbb{F}_p$ by adjusting the result to the correct powers (respectively $p - 2$, $\frac{p-1}{2}$, $\frac{p+1}{4}$). In the optimized code, we implemented Pornin's algorithm for inversion [Por20a] and the Legendre symbol [Por20b] in constant-time, so they could be used for both signature and verification.

In terms of cost, inverting in $\mathbb{F}_{p^2}$ amounts to computing an inversion, two multiplications, two squarings, and a few additions in $\mathbb{F}_p$. Quadratic character computations in $\mathbb{F}_{p^2}$ require two squarings, one addition, and a Legendre symbol in $\mathbb{F}_p$. For square root computation, only two exponentiations and a few multiplications in $\mathbb{F}_p$ are required, as shown in Algorithm 8.2.

8.2. Elliptic curve arithmetic

This section describes explicit formulas for the curve arithmetic using Montgomery curves, with both projective x-only coordinates and projective coordinates.

Algorithm 8.2 SquareRoot(a)

Input: A quadratic residue $a = a_0 + ia_1 \in \mathbb{F}_{p^2}$, with $i^2 = -1$.
Output: A deterministically chosen element $x \in \mathbb{F}_{p^2}$ such that $x^2 = a$.

$\delta \leftarrow (a_0^2 + a_1^2)^{(p+1)/4}$
$x_0 \leftarrow a_0 + \delta$
$t_0 \leftarrow 2x_0$
$x_1 \leftarrow t_0^{(p-3)/4}$
$x_0 \leftarrow x_0 x_1$
$x_1 \leftarrow a_1 x_1$
$t_1 \leftarrow (2x_0)^2$
if $t_1 = t_0$ **then**
    return $x_0 + ix_1$
else
    return $x_1 - ix_0$

8.2.1. Projective x-only coordinates

Let E be a Montgomery curve defined over $\mathbb{F}_q$ by the equation $By^2 = x^3 + Ax^2 + x$ with $A, B \in \mathbb{F}_q$. The points of E over $\mathbb{F}_q$ are uniquely represented by their affine coordinates (x, y). In our implementation, we work with *projective coordinates* $(X : Y : Z)$ instead, where points of the curve satisfy

$$BY^2Z = X^3 + AX^2Z + XZ^3$$

and X, Y, Z are not simultaneously zero. Projective points are defined up to scalar multiplication, which means that we identify $(\lambda X : \lambda Y : \lambda Z)$ with $(X : Y : Z)$ for all $\lambda \in \mathbb{F}_q^*$. When $Z \neq 0$, the projective point $P = (X : Y : Z)$ corresponds to the affine point $(x, y) = (X/Z, Y/Z)$. When $Z = 0$, the projective point $P = (0 : 1 : 0) = 0_E$ is the *point at infinity*, which does not have an affine representation.

For efficiency reasons, we use x-only or $(X : Z)$-only arithmetic, disregarding the y- or Y-coordinate completely. With these coordinates, points are determined only up to sign, which is enough for our context. Namely, the coordinates $(X : Z)$ correspond to the two points $P = (X : Y : Z)$ and $-P = (X : -Y : Z)$ on E.

Additionally, our implementation disregards the curve parameter B which only affects the y-coordinates of points. We represent A projectively as $A = (A_{\text{Pr}} : C_{\text{Pr}})$ with $A = A_{\text{Pr}}/C_{\text{Pr}}$, and will often use the alternative representation $(A_{24} : C_{24}) = (A_{\text{Pr}} + 2C_{\text{Pr}} : 4C_{\text{Pr}})$.

8.2.2. Curve arithmetic

Below we give pseudocode for the x-only operations that our implementation uses. The basic operations are the following:

- xDBL (Algorithm 8.3) takes as input the projective x-coordinate of a point P and the projective curve parameters $(A_{24} : C_{24})$, and outputs the projective x-coordinate of the point $[2]P$.
- xADD (Algorithm 8.4) takes as input the projective x-coordinates of the points P, Q, and $P - Q$, and outputs the projective x-coordinate of the point $P + Q$.
- xDBLADD (Algorithm 8.5) combines an instance of xDBL and xADD in an efficient way. It takes as input the projective x-coordinates of the points P, Q, $P - Q$, and the projective curve parameters $(A_{24} : C_{24})$, and outputs the projective x-coordinates of the points $P + Q$ and $[2]P$.

Using these basic operations, we can describe the following variants of scalar multiplications:

- Ladder (Algorithm 8.6) takes as input the projective x-coordinate of the point P, the projective curve parameters $(A_{24} : C_{24})$, and a positive scalar m, and outputs the projective x-coordinate of the point $[m]P$.
- Ladder3pt (Algorithm 8.7) takes as input the projective x-coordinates of the points P, Q, $P - Q$, the projective curve parameters $(A_{24} : C_{24})$, and a positive scalar m, and outputs the projective x-coordinate of the point $P + [m]Q$.
- LadderBiscalar (Algorithm 8.8) takes as input the projective x-coordinates of the points P, Q, $P - Q$, the projective curve parameters $(A_{24} : C_{24})$, and positive scalars m and n, and outputs the projective x-coordinate of the point $[m]P + [n]Q$.

We also describe IsomorphismMontgomeryCurves (Algorithm 8.9), which is a projective x-only implementation of Equation (6) to compute an isomorphism between two curves and push points through it.

Algorithm 8.3 $\mathsf{xDBL}(P, (A_{24} : C_{24}))$

Input: A projective point $P = (X_P : Z_P)$ and the Montgomery coefficient $(A_{24} : C_{24})$ of the curve E.
Output: The projective point $[2]P = (X_{2P} : Z_{2P})$.

1: $t_0 \leftarrow X_P + Z_P$
2: $t_0 \leftarrow t_0^2$
3: $t_1 \leftarrow X_P - Z_P$
4: $t_1 \leftarrow t_1^2$
5: $t_2 \leftarrow t_0 - t_1$
6: $t_1 \leftarrow t_1 \cdot C_{24}$
7: $X_{2P} \leftarrow t_0 \cdot t_1$
8: $t_0 \leftarrow t_2 \cdot A_{24}$
9: $t_0 \leftarrow t_0 + t_1$
10: $Z_{2P} \leftarrow t_0 \cdot t_2$
11: **return** $[2]P = (X_{2P} : Z_{2P})$ // Cost: $2\mathbf{S} + 4\mathbf{M} + 4\mathbf{a}$

Algorithm 8.4 $\mathsf{xADD}(P, Q, P - Q)$

Input: Projective points $P = (X_P : Z_P), Q = (X_Q : Z_Q), P - Q = (X_{P-Q} : Z_{P-Q})$.
Output: The projective sum $P + Q = (X_{P+Q} : Z_{P+Q})$.

1: $t_0 \leftarrow X_P + Z_P$
2: $t_1 \leftarrow X_P - Z_P$
3: $t_2 \leftarrow X_Q + Z_Q$
4: $t_3 \leftarrow X_Q - Z_Q$
5: $t_0 \leftarrow t_0 \cdot t_3$
6: $t_1 \leftarrow t_1 \cdot t_2$
7: $t_2 \leftarrow t_0 + t_1$
8: $t_3 \leftarrow t_0 - t_1$
9: $t_2 \leftarrow t_2^2$
10: $t_3 \leftarrow t_3^2$
11: $X_{P+Q} \leftarrow Z_{P-Q} \cdot t_2$
12: $Z_{P+Q} \leftarrow X_{P-Q} \cdot t_3$
13: **return** $P + Q = (X_{P+Q} : Z_{P+Q})$ // Cost: $2\mathbf{S} + 4\mathbf{M} + 6\mathbf{a}$

Algorithm 8.5 $\mathsf{xDBLADD}(P, Q, P-Q, (A_{24} : C_{24}))$

Input: Projective points $P = (X_P : Z_P), Q = (X_Q : Z_Q), P - Q = (X_{P-Q} : Z_{P-Q})$. and the Montgomery coefficient $(A_{24} : C_{24})$ of the curve E.
Output: Projective points $[2]P = (X_{2P} : Z_{2P})$ and $P + Q = (X_{P+Q} : Z_{P+Q})$.

$t_0 \leftarrow X_P + Z_P$
$t_1 \leftarrow X_P - Z_P$
$X_{2P} \leftarrow t_0^2$
$t_2 \leftarrow X_Q - Z_Q$
$X_{P+Q} \leftarrow X_Q + Z_Q$
$t_0 \leftarrow t_0 \cdot t_2$
$Z_{2P} \leftarrow t_1^2$
$t_1 \leftarrow t_1 \cdot X_{P+Q}$
$t_2 \leftarrow X_{2P} - Z_{2P}$
$Z_{2P} \leftarrow Z_{2P} \cdot C_{24}$
$X_{2P} \leftarrow X_{2P} \cdot Z_{2P}$
$X_{P+Q} \leftarrow A_{24} \cdot t_2$
$Z_{P+Q} \leftarrow t_0 - t_1$
$Z_{2P} \leftarrow Z_{2P} + X_{P+Q}$
$X_{P+Q} \leftarrow t_0 + t_1$
$Z_{2P} \leftarrow Z_{2P} \cdot t_2$
$Z_{P+Q} \leftarrow Z_{P+Q}^2$
$X_{P+Q} \leftarrow X_{P+Q}^2$
$Z_{P+Q} \leftarrow Z_{P+Q} \cdot X_{P-Q}$
$X_{P+Q} \leftarrow X_{P+Q} \cdot Z_{P-Q}$
return $([2]P, P+Q) = ((X_{2P} : Z_{2P}), (X_{P+Q} : Z_{P+Q}))$ // Cost: $4\mathbf{S} + 8\mathbf{M} + 8\mathbf{a}$

Algorithm 8.6 $\mathsf{Ladder}(P, E, m)$

Input: A projective point $P = (X_P : Z_P)$, the Montgomery coefficient $(A_{24} : C_{24})$ of the curve E, and a positive scalar m with binary representation $m = (m_{k-1}, \ldots, m_0)_2$.
Output: The projective point $[m]P = (X_{[m]P} : Z_{[m]P})$.

$((X_0 : Z_0), (X_1 : Z_1)) \leftarrow ((1 : 0), (X_P : Z_P))$
for i **from** $k-1$ **down to** 0 **do**
  if $m_i = 1$ **then**
    $((X_1, Z_1), (X_0, Z_0)) \leftarrow \mathsf{xDBLADD}((X_1 : Z_1), (X_0 : Z_0), (X_P : Z_P), (A_{24} : C_{24}))$
  else
    $((X_0 : Z_0), (X_1 : Z_1)) \leftarrow \mathsf{xDBLADD}((X_0 : Z_0), (X_1 : Z_1), (X_P : Z_P), (A_{24} : C_{24}))$
$(X_{[m]P} : Z_{[m]P}) \leftarrow (X_0 : Z_0)$
return $[m]P = (X_{[m]P} : Z_{[m]P})$

Algorithm 8.7 $\mathsf{Ladder3pt}(P, Q, P-Q, (A_{24} : C_{24}), m)$

Input: Projective points $P = (X_P : Z_P), Q = (X_Q : Z_Q), P - Q = (X_{P-Q} : Z_{P-Q})$, the Montgomery coefficient $(A_{24} : C_{24})$ of the curve E, and a positive scalar m with binary representation $m = (m_{k-1}, \ldots, m_0)_2$.
Output: The projective point $P + [m]Q = (X_{P+[m]Q} : Z_{P+[m]Q})$.

$((X_0 : Z_0), (X_1 : Z_1), (X_2 : Z_2)) \leftarrow ((X_P : Z_P), (X_Q : Z_Q), (X_{P-Q} : Z_{P-Q}))$
for i **from** 0 **up to** $k-1$ **do**
  if $m_i = 1$ **then**
    $((X_0, Z_0), (X_1, Z_1)) \leftarrow \mathsf{xDBLADD}((X_0 : Z_0), (X_1 : Z_1), (X_2 : Z_2), (A_{24} : C_{24}))$
  else
    $((X_0 : Z_0), (X_2 : Z_2)) \leftarrow \mathsf{xDBLADD}((X_0 : Z_0), (X_2 : Z_2), (X_1 : Z_1), (A_{24} : C_{24}))$
$(X_{P+[m]Q} : Z_{P+[m]Q}) \leftarrow (X_1 : Z_1)$
return $P + [m]Q = (X_{P+[m]Q} : Z_{P+[m]Q})$

Algorithm 8.8 LadderBiscalar$(P, Q, P-Q, (A_{24} : C_{24}), m, n)$

Input: Projective points $P = (X_P : Z_P), Q = (X_Q : Z_Q), P - Q = (X_{P-Q} : Z_{P-Q})$, the Montgomery coefficient $(A_{24} : C_{24})$ of the curve E, and positive scalars m, n with binary representations $m = (m_{k-1}, \ldots, m_0)_2$ and $n = (n_{k-1}, \ldots, n_0)_2$, resp.
Output: The projective point $[m]P + [n]Q = (X_{[m]P+[n]Q} : Z_{[m]P+[n]Q})$.

Recoding stage:
if m is even **and** n is odd **then**
  $\sigma = (\sigma_0, \sigma_1) \leftarrow (1, 0)$
else
  $\sigma = (\sigma_0, \sigma_1) \leftarrow (0, 1)$
$m' \leftarrow m, n' \leftarrow n$
if m is even **then**
  $m' \leftarrow m' - 1$
if n is even **then**
  $n' \leftarrow n' - 1$
Set $b = (b_0, b_1)$, where $b_0 \leftarrow (0, m'_{k-1}, \ldots, m'_0)$ and $b_1 \leftarrow (0, n'_{k-1}, \ldots, n'_0)$
for i **from** 0 **up to** $k - 1$ **do**
  $r_{2i} \leftarrow b_{\sigma_0,i} \oplus b_{\sigma_0,i+1}$
  $r_{2i+1} \leftarrow b_{\sigma_1,i} \oplus b_{\sigma_1,i+1}$
  if $r_{2i+1} = 1$ **then**
    $t \leftarrow \sigma_0$
    $\sigma_0 \leftarrow \sigma_1$
    $\sigma_1 \leftarrow t$
Evaluation stage:
$R_0 \leftarrow (1 : 0)$
$T = (T_0, T_1) \leftarrow (P, Q)$
$R_1 \leftarrow T_{\sigma_0}$
$R_2 \leftarrow T_{\sigma_0+1 \pmod 2}$
$D_1 \leftarrow R_1$
$D_2 \leftarrow R_2$
$R_2 \leftarrow \mathsf{xADD}(R_1, R_2, P - Q)$
$F_1 \leftarrow R_2$
$F_2 \leftarrow P - Q$
for i **from** $k - 1$ **down to** 0 **do**
  $h \leftarrow r_{2i} + r_{2i+1}$
  $T_0 \leftarrow R_{h \pmod 2}$
  $T = (T_0, T_1) \leftarrow (T_0, R_2)$
  $T_0 \leftarrow \mathsf{xDBL}(T_{\lfloor h/2 \rfloor}, (A_{24} : C_{24}))$
  $T_1 \leftarrow R_{r_{2i+1}}$
  $T_2 \leftarrow R_{r_{2i+1}+1}$
  if $r_{2i+1} = 1$ **then**
    $TMP \leftarrow D_1$
    $D_1 \leftarrow D_2$
    $D_2 \leftarrow TMP$
  $T_1 \leftarrow \mathsf{xADD}(T_1, T_2, D_1)$
  $T_2 \leftarrow \mathsf{xADD}(R_0, R_2, F_1)$
  if $h \pmod 2 = 1$ **then**
    $TMP \leftarrow F_1$
    $F_1 \leftarrow F_2$
    $F_2 \leftarrow TMP$
  $R_0 \leftarrow T_0$
  $R_1 \leftarrow T_1$
  $R_2 \leftarrow T_2$
$(X_{[m]P+[n]Q} : Z_{[m]P+[n]Q}) \leftarrow R_{(m \pmod 2 \oplus 1) + (n \pmod 2 \oplus 1)}$
return $[m]P + [n]Q = (X_{[m]P+[n]Q} : Z_{[m]P+[n]Q})$

Algorithm 8.9 IsomorphismMontgomeryCurves(E, P, Q, E')

Input: Montgomery coefficients $(A : C)$ and $(A' : C')$ of the curves E and E', respectively, and points $P = (X_P : Z_P), Q = (X_Q : Z_Q)$ in E.
Output: The images $P' = (X'_P : Z'_P), Q' = (X'_Q : Z'_Q)$ of P and Q under an isomorphism between E and E'.

$\lambda_x \leftarrow (2A'^3 - 9A'C'^2)(3C^3 - A^2C)$
$\lambda_z \leftarrow (2A^3 - 9AC^2)(3C'^3 - A'^2C')$
if $\lambda_x = 0$ **or** $\lambda_z = 0$ **then**
  raise ("IsomorphismMontgomeryCurves: invalid input curve.")
$X_{P'} \leftarrow \lambda_x(3X_PCC' + AC'Z_P) - \lambda_z A'CZ_P$
$Z_{P'} \leftarrow 3\lambda_z CC'Z_P$
$X_{Q'} \leftarrow \lambda_x(3X_QCC' + AC'Z_Q) - \lambda_z A'CZ_Q$
$Z_{Q'} \leftarrow 3\lambda_z CC'Z_Q$
return $(X_{P'} : Z_{P'}), (X_{Q'} : Z_{Q'})$

REMARK 1. Algorithm 8.9 fails for the specific cases when either $(A/C)^2$ or $(A'/C')^2$ is in $\{0, 3, 9/2\}$, which are corner cases that arise from writing Equation (6) projectively. This introduces a probability of failure that is negligible as long as the inputs are honestly generated, which does not represent a problem since the algorithm is only used in signing.

8.2.3. Projective point difference

Recall that Equation (7) forces us to make an arbitrary choice of sign when computing the difference of two points using x-only coordinates. While either of the choices is valid, it is important that the choice be done deterministically so that the same basis is obtained during signing and verification. Using projective coordinates introduces a new challenge, since we must specify the behavior of the square root on a projective quantity in a way that is projectively invariant. For instance, a natural choice that requires only one square root over $\mathbb{F}_{p^2}$ is to define $\sqrt{(x : z)} := (\sqrt{xz} : z)$, but this definition is not projectively invariant (that is, $\sqrt{(x : z)}$ and $\sqrt{(\lambda x : \lambda z)}$ may in general differ by a sign). Instead, we define

$$\sqrt{(x : z)} := \left(\sqrt{xz^3\bar{z}^4} : z^2\bar{z}^2\right).$$

This definition can be shown to be projectively invariant, thanks to the fact that our choice of square root over $\mathbb{F}_{p^2}$ satisfies the special property that $\sqrt{\lambda^4 x} = \lambda^2\sqrt{x}$ for all $x \in \mathbb{F}_{p^2},\ \lambda \in \mathbb{F}_p$.

The ProjectiveDifference algorithm implements Equation (7) for r_+ projectively using this principle.

Algorithm 8.10 ProjectiveDifference$(P, Q, (A : C))$

Input: Projective points $P = (X_P : Z_P)$ and $Q = (X_Q : Z_Q)$ and the Montgomery coefficient $(A : C)$.
Output: A deterministic x-coordinate x_{PQ}, either x_{P-Q} or x_{P+Q}.

$B_{XX} \leftarrow C \cdot (X_PX_Q - Z_PZ_Q)^2$
$B_{XZ} \leftarrow C \cdot (X_PX_Q + Z_PZ_Q)(X_PZ_Q + Z_PX_Q) + 2AX_PX_QZ_PZ_Q$
$B_{ZZ} \leftarrow C \cdot (X_PZ_Q - Z_PX_Q)^2$
$\gamma \leftarrow C \cdot (\overline{C} \cdot \overline{Z_P} \cdot \overline{Z_Q})^2$
$B_{XX} \leftarrow \gamma \cdot B_{XX},\ B_{XZ} \leftarrow \gamma \cdot B_{XZ},\ B_{ZZ} \leftarrow \gamma \cdot B_{ZZ},$
$\delta \leftarrow$ SquareRoot$(B_{XZ}^2 - B_{XX}B_{ZZ})$
$X_{PQ} \leftarrow \delta + B_{XZ},\ Z_{PQ} \leftarrow B_{ZZ}$
return $x_{PQ} = (X_{PQ}, Z_{PQ})$

8.2.4. Jacobian coordinates

Our implementation occasionally requires full projective coordinate arithmetic instead of just x-only. This is, for example, the case in Section 8.5.5. In such cases, for efficiency reasons, we define our arithmetic using Jacobian coordinates, in which a point P over the curve E is defined by coordinates $(X : Y : Z)$ satisfying

$$BY^2 = X^3 + AX^2Z^2 + XZ^4$$

with the point at infinity $(0:1:0)$. Instead of the standard projective equivalence $(X:Y:Z) \sim (\lambda X:\lambda Y:\lambda Z)$ for any $\lambda \in \mathbb{F}_q^*$, the Jacobian equivalence is defined as $(X:Y:Z) \sim (\lambda^2 X:\lambda^3 Y:\lambda Z)$ for any $\lambda \in \mathbb{F}_q^*$. The transformation between Montgomery and Jacobian coordinates is defined as follows:

- A point P in Jacobian coordinates $(X_J:Y_J:Z_J)$ corresponds to Montgomery coordinates

$$(X_M:Y_M:Z_M) = \left(X_J:Y_J/Z_J:Z_J^2\right).$$

- Conversely, a point P in Montgomery coordinates $(X_M:Y_M:Z_M)$ has Jacobian representation

$$(X_J:Y_J:Z_J) = \left(X_M - AZ_M^2/3, Y_M, Z_M\right).$$

Below, we give pseudocode for arithmetic in Jacobian coordinates. The basic operations are the following:

- DBL (Algorithm 8.11) takes as input the Jacobian coordinates of a point P and the projective curve parameters $(A_{24}:C_{24})$, and outputs the Jacobian coordinates of the point $[2]P$.
- ADD (Algorithm 8.12) constant time, complete addition formula, takes as input the Jacobian coordinates of two points P, Q and the projective curve parameters $(A_{24}:C_{24})$, and outputs the Jacobian coordinates of the point $P+Q$.
- ADDComponents (Algorithm 8.13) takes as input the Jacobian coordinates of distinct points P, Q and the projective curve parameters $(A_{24}:C_{24})$, and outputs u, v, w, that describe the projective x-only coordinates of $P+Q$ and $P-Q$, with $P+Q=(u-v:w)$ and $P-Q=(u+v:w)$.

Algorithm 8.11 DBL$(P,(A_{24}:C_{24}))$

Input: Jacobian point $P=(X_P:Y_P:Z_P)$ and the Montgomery coefficient $(A_{24}:C_{24})$ of the curve E.
Output: The Jacobian point $[2]P=(X_{[2]P}:Y_{[2]P}:Z_{[2]P})$.

1: $t_0 \leftarrow X_P^2$
2: $t_1 \leftarrow t_0 + t_0$
3: $t_0 \leftarrow t_0 + t_1$
4: $t_1 \leftarrow Z_P^2$
5: $t_2 \leftarrow X_P \cdot A_{24}$
6: $t_2 \leftarrow t_2 + t_2$
7: $t_2 \leftarrow t_1 + t_2$
8: $t_2 \leftarrow t_1 \cdot t_2$
9: $t_2 \leftarrow t_0 + t_2$
10: $Z_{[2]P} \leftarrow Y_P \cdot Z_P$
11: $Z_{[2]P} \leftarrow Z_{[2]P} + Z_{[2]P}$
12: $t_0 \leftarrow Z_{[2]P}^2$
13: $t_0 \leftarrow t_0 \cdot A_{24}$
14: $t_1 \leftarrow Y_P^2$
15: $t_1 \leftarrow t_1 + t_1$
16: $t_3 \leftarrow x_P + x_P$
17: $t_3 \leftarrow t_1 \cdot t_3$
18: $X_{[2]P} \leftarrow t_2^2$
19: $X_{[2]P} \leftarrow X_{[2]P} - t_0$
20: $X_{[2]P} \leftarrow X_{[2]P} - t_3$
21: $X_{[2]P} \leftarrow X_{[2]P} - t_3$
22: $Y_{[2]P} \leftarrow t_3 - X_{[2]P}$
23: $Y_{[2]P} \leftarrow Y_{[2]P} \cdot t_2$
24: $t_1 \leftarrow t_1^2$
25: $Y_{[2]P} \leftarrow Y_{[2]P} - t_1$
26: $Y_{[2]P} \leftarrow Y_{[2]P} - t_1$
27: **return** $[2]P=(X_{[2]P}:Y_{[2]P}:Z_{[2]P})$ // Cost: $6\mathbf{S}+6\mathbf{M}+14\mathbf{a}$

Algorithm 8.12 $\mathsf{ADD}(P, Q, (A_{24} : C_{24}))$

Input: Two Jacobian points $P = (X_P : Y_P : Z_P)$, $Q = (X_Q : Y_Q : Z_Q)$ and the Montgomery coefficient $(A_{24} : C_{24})$ of the curve E.
Output: The Jacobian point $P + Q = (X_{P+Q} : Y_{P+Q} : Z_{P+Q})$.

1: $\mathsf{ctl}_1 \leftarrow (P = 0)$
2: $\mathsf{ctl}_2 \leftarrow (Q = 0)$
3: $t_0 \leftarrow Z_P^2$
4: $t_1 \leftarrow Z_Q^2$
5: $v_1 \leftarrow t_1 \cdot Z_Q$
6: $t_2 \leftarrow t_0 \cdot Z_P$
7: $v_1 \leftarrow v_1 \cdot Y_P$
8: $t_2 \leftarrow t_2 \cdot Y_Q$
9: $dy \leftarrow t_2 - v_1$
10: $u_2 \leftarrow t_0 \cdot X_Q$
11: $u_1 \leftarrow t_1 \cdot X_P$
12: $dx \leftarrow u_2 - u_1$
13: $t_1 \leftarrow Y_P + Y_P$
14: $t_2 \leftarrow A_{24} \cdot X_P$
15: $t_2 \leftarrow t_2 + t_2$
16: $t_2 \leftarrow t_2 + t_0$
17: $t_2 \leftarrow t_2 \cdot t_0$
18: $t_0 \leftarrow X_P^2$
19: $t_2 \leftarrow t_2 + t_0$
20: $t_2 \leftarrow t_2 + t_0$
21: $t_2 \leftarrow t_2 + t_0$
22: $t_2 \leftarrow t_2 \cdot Z_Q$
23: $\mathsf{ctl} \leftarrow (dx = 0) \wedge (dy = 0)$
24: **if** ctl **then** swap (dx, t_1) and (dy, t_2)
25: $t_0 \leftarrow Z_P \cdot Z_Q$
26: $t_1 \leftarrow t_0^2$
27: $t_2 \leftarrow dx^2$
28: $t_3 \leftarrow dy^2$
29: $X_{P+Q} \leftarrow A_{24} \cdot t_1$
30: $X_{P+Q} \leftarrow X_{P+Q} + u_1$
31: $X_{P+Q} \leftarrow X_{P+Q} + u_2$
32: $X_{P+Q} \leftarrow X_{P+Q} \cdot t_2$
33: $X_{P+Q} \leftarrow t_3 - X_{P+Q}$
34: $Y_{P+Q} \leftarrow u_1 \cdot t_2$
35: $Y_{P+Q} \leftarrow Y_{P+Q} - X_{P+Q}$
36: $Y_{P+Q} \leftarrow dy \cdot Y_{P+Q}$
37: $t_3 \leftarrow dx \cdot t_2$
38: $t_3 \leftarrow t_3 \cdot v_1$
39: $Y_{P+Q} \leftarrow Y_{P+Q} - t_3$
40: $Z_{P+Q} \leftarrow dx \cdot t_0$
41: **if** ctl_1 **then** set $R \leftarrow Q$
42: **if** ctl_2 **then** set $R \leftarrow P$
43: **return** $P + Q = (X_{P+Q} : Y_{P+Q} : Z_{P+Q})$ // Cost: $6\mathbf{S} + 17\mathbf{M} + 13\mathbf{a}$

Algorithm 8.13 ADDComponents$(P, Q, (A_{24} : C_{24}))$

Input: Distinct Jacobian points $P = (X_P : Y_P : Z_P)$, $Q = (X_Q : Y_Q : Z_Q)$ and the Montgomery coefficient $(A_{24} : C_{24})$ of the curve E.
Output: (u, v, w) such that the x-only coordinates of $P + Q$ and $P - Q$ are $P + Q = (u - v : w)$ and $P - Q = (u + v : w)$.

1: $t_0 \leftarrow Z_P^2$
2: $t_1 \leftarrow Z_Q^2$
3: $t_2 \leftarrow X_P \cdot t_1$
4: $t_3 \leftarrow t_0 \cdot X_Q$
5: $t_4 \leftarrow Y_P \cdot Z_Q$
6: $t_4 \leftarrow t_4 \cdot t_1$
7: $t_5 \leftarrow Z_P \cdot Y_Q$
8: $t_5 \leftarrow t_5 \cdot t_0$
9: $t_0 \leftarrow t_0 \cdot t_1$
10: $t_6 \leftarrow t_4 \cdot t_5$
11: $t_4 \leftarrow t_4^2$
12: $t_5 \leftarrow t_5^2$
13: $t_4 \leftarrow t_4 + t_5$
14: $t_5 \leftarrow t_2 + t_3$
15: $t_7 \leftarrow t_3 + t_3$
16: $t_7 \leftarrow t_5 - t_7$
17: $t_7 \leftarrow t_7^2$
18: $t_1 \leftarrow A_{24} \cdot t_0$
19: $t_1 \leftarrow t_5 + t_1$
20: $t_1 \leftarrow t_1 \cdot t_7$
21: $u \leftarrow t_4 - t_1$
22: $v \leftarrow t_6 + t_6$
23: $w \leftarrow t_6 \cdot t_0$
24: **return** (u, v, w) // Cost: $5\mathbf{S} + 11\mathbf{M} + 7\mathbf{a}$

8.3. Computing pairings

This section describes the computation of pairings in the reference implementation of SQIsign. Usage of pairings is described in Section 2.2.5. There are several ways to compute $T_n(P, Q)$ given P, Q and n. The most popular methods include Miller's loop approach [Mil04], cubical arithmetic [Rob24], and elliptic nets [Sta07]. In the remaining of this section, we outline the first two methods.

In general, a Miller function $f_{n,P}$ is a function $f : E \to \mathbb{P}^1$ with a zero of multiplicity n at P, a pole at $[n]P$ and a pole with multiplicity $n-1$ at 0_E, that is, $\mathrm{Div}(f_{n,P}) = n(P) - ([n]P) - (n-1)(0_E)$. If $P \in E[n]$, this reduces to $\mathrm{Div}(f_{n,P}) = n(P) - n(0_E)$. Defining divisors $D_P = (P) - (0_E)$ and $D_Q = (Q) - (0_E)$, and given such functions $f_{n,P}$ and $f_{n,Q}$, the (unreduced) Tate pairing for $P, Q \in E[n]$ can be effectively computed as $T_n(P, Q) = f_{n,P}(D_Q)$. With the exception of minor technical details, which are thoroughly discussed in [BKLS02], we can simply compute $f_{n,P}(D_Q) = f_{n,P}(Q)$.

8.3.1. The Miller loop approach

This approach has been intensively studied in the literature (see [FST10] for a survey of pairings, and [Rei23] for some applications of pairings in isogeny-based cryptography). In a nutshell, T_n can be obtained by first computing a Miller loop that computes the Miller function f_{nP}, satisfying $\mathrm{div}(f_{nP}) = n(P) - n(0_E)$ [Mil04]. The Miller loop computes f_{nP} using a double-and-add algorithm with $\lceil \log_2(n) \rceil$ iterations. More specifically, the value of f is updated at each iteration according to the Miller formula,

$$f_{(n+m)P} = f_{nP} \cdot f_{mP} \cdot \frac{l}{v}$$

where l and v are the lines associated to the addition of the points nP and mP. When $n = m$, we refer to this as doubling nP, and when $m = 1$, we add the points nP and P. SQIsign only requires pairings of degree $n = 2^e$, hence we only need to perform exactly e doublings and no additions in the Miller loop. Having computed $f_{nP}(Q)$, we raise it to the power

$$\frac{q-1}{n} = (p-1) \cdot \frac{p+1}{n},$$

to compute the unreduced Tate pairing t_n. Using the Frobenius operator $\pi : x \to x^p$ which acts on $\mathbb{F}_{p^2}$ as $\pi(a + bi) = a - bi$, and the nearly free divisions of elements over the subgroup μ_n, we can easily exponentiate $f_{nP}(Q)$ to the power $p-1$. Finally, since $\frac{p+1}{n}$ is a small integer, this final exponentiation is relatively inexpensive. This completes the computation of the Tate pairing using the Miller approach presented in [Mil04].

Although this is by far the most popular approach for computing pairings, the Miller loop has generally been defined using affine or projective coordinates, rather than in the context of x-only elliptic-curve arithmetic, which is the preferred approach in SQIsign. Cubical arithmetic addresses this limitation.

8.3.2. The cubical arithmetic approach

In [Sta08], the author explains that $T_n(P, Q)$ can be reinterpreted as the monodromy $g^{\star_1,n}_{P,Q} = T_n(P, Q)$, viewed as a constant function on the curve, in the biextension associated to the principal polarization (0_E) on E, where $g_{P,Q}$ is a rational biextension element above (P, Q). This idea is further developed in [Rob24], where it is shown that the biextension arithmetic and the associated monodromy can be refined by means of cubical arithmetic.

We note that will be working with the cubical arithmetic of level 2, associated to the divisor $2(0_E)$, on an elliptic curve in Montgomery form $E : y^2 = x^3 + Ax^2 + 1$, using the cubical formulas from [Rob24]. We have $\Gamma(2(0_E)) = (X, Z)$, where $x = X/Z$ represent the x-Weierstrass coordinate of E. The x coordinate $x(P)$ of a point $P = (x(P), y(P)) \in E$ allows to recover P up to a sign. We can also use the projective coordinates $(X(P) : Z(P)) = (x(P) : 1)$. A (level 2) cubical point $\tilde{P}$ above P is specified by a choice $\tilde{P} = (X(P), Z(P))$ of affine coordinates such that $x(P) = X(P)/Z(P)$.

Using cubical arithmetic, computing $T_n(P, Q)$ requires us to compute two monodromies λ_P and $\lambda_{P,Q}$, where $\lambda_P \in \mathbb{F}_{p^2}$ such that $\widetilde{[n]P} = \lambda_P \cdot \widetilde{0}$ and $\lambda_{P,Q} \in \mathbb{F}_{p^2}$ such that $\widetilde{[n]P + Q} = \lambda_{P,Q} \cdot \widetilde{Q}$, with $\widetilde{0} = (0, 1)$, $\widetilde{Q} = (x_Q, 1)$, where $\widetilde{[n]P}$ and $\widetilde{[n]P + Q}$ are the outputs of CubicalLadder, given P, Q and n. We then find $T_n(P, Q) = \lambda_{P,Q}/\lambda_P$. This is the result of the Monodromy algorithm.

When $n \mid p+1$, the *final reduction* to obtain $t_n(P, Q)$ requires us to raise $T_n(P, Q)$ to $(p^2-1)/n = (p-1) \cdot \frac{p+1}{n}$. Raising $\zeta \in \mathbb{F}_{p^2}$ to the power $(p-1)$ is again simply $\pi(\zeta)/\zeta$. Hence, the final reduction can be computed as

$$t_n(P, Q) = (\pi(\lambda_{P,Q})/\pi(\lambda_P) \cdot \lambda_P/\lambda_{P,Q})^{\frac{p+1}{n}}. \tag{10}$$

The Tate algorithm computes this value $t_n(P, Q)$ given P and Q.

To summarize, Algorithm 8.19 computes the Tate algorithm by first calculating a monodromy λ followed by the final exponentiation to the power $\frac{p^2-1}{2^e}$. The monodromy is computed by executing the three-point cubical arithmetic ladder in Algorithm 8.16, which uses the cubical double and cubical differential additions from Algorithm 8.14 and Algorithm 8.15 as auxiliary blocks. Notice also that the monodromy computation requires performing a couple of cubical translations from Algorithm 8.17 and ratios from Algorithm 8.18.

We warn the reader that our implementation of the Tate algorithm is slightly different from the simplified description given above, as we use x-only arithmetic. Thus, we work with cubical points of level 2, which encode elements in the biextension associated to the divisor $2(0_E)$ rather than (0_E). The former biextension monodromy yields the square of the Tate pairing $T_n(P, Q)$. Since we will use the Tate pairing for n a power of two, this loses one bit of information. Instead, we use the action of the theta group $G(2(0_E))$, via CubicalTranslate, to get the correct monodromy factor, i.e., from the biextension associated to (0_E).

A final important caveat is that CubicalDiffAdd as described is off by a factor 4 from the correct cubical arithmetic: in the algorithm we should return $(X_2/4, Z_2/4)$. However, the current version still gives the correct Weil pairing $e_n(P, Q) = T_n(P, Q)/T_n(Q, P)$, and, thanks to the final exponentiation, the reduced Tate-Lichtenbaum pairing $t_n(P, Q)$ over $\mathbb{F}_{p^2}$. But one should be careful that the current version will not give the correct reduced Tate-Lichtenbaum pairing if computed over $\mathbb{F}_p$. Note that SQIsign only explicitly computes the Tate pairing.

Algorithm 8.14 CubicalDbl$(E, \widetilde{P})$

Input: An elliptic curve $E : y^2 = x^3 + Ax^2 + x$ in Montgomery model and a cubical point $\widetilde{P} = (X(P), Z(P))$.
Output: The cubical double $2\widetilde{P}$.

1: $a \leftarrow (X(P) + Z(P))^2$
2: $b \leftarrow (X(P) - Z(P))^2$
3: $c \leftarrow a - b$
4: $X_2 \leftarrow a \cdot b$
5: $Z_2 \leftarrow c \cdot (b + \frac{A+2}{4} \cdot c)$
6: **return** (X_2, Z_2)

Algorithm 8.15 CubicalDiffAdd$(E, \widetilde{P}, \widetilde{Q}, x(P - Q))$

Input: Two cubical points in cubical Montgomery coordinates $\widetilde{P} = (X(P), Z(P)), \widetilde{Q} = (X(Q), Z(Q))$ and the x-coordinate of their difference $x(P - Q)$.
Output: The cubical differential addition $\widetilde{P + Q}$.

1: $a \leftarrow X(P) + Z(P)$
2: $b \leftarrow X(P) - Z(P)$
3: $c \leftarrow X(Q) + Z(Q)$
4: $d \leftarrow X(Q) - Z(Q)$
5: $X_2 \leftarrow (a \cdot d + b \cdot c)^2$
6: $Z_2 \leftarrow (a \cdot d - b \cdot c)^2$
7: $X_2 \leftarrow X_2 / x(P - Q)$
8: **return** (X_2, Z_2)

Algorithm 8.16 CubicalLadder$(E, e, \widetilde{P + Q}, \widetilde{P}, x(Q))$

Input: An elliptic curve $E : y^2 = x^3 + Ax^2 + x$ in Montgomery model, an integer e, two cubical points $\widetilde{P + Q} = (X(P + Q), Z(P + Q)), \widetilde{P} = (X(P), Z(P))$ and the x-coordinate $x(Q)$ of Q.
Output: The cubical points $2^e\widetilde{P}$ and $2^e\widetilde{P} + \widetilde{Q}$.

1: $nPQ \leftarrow \widetilde{P + Q}$
2: $nP \leftarrow \widetilde{P}$
3: **for** k **from** 1 **up to** e **do**
4: $\quad nPQ \leftarrow$ CubicalDiffAdd$(E, nPQ, nP, x(Q))$
5: $\quad nP \leftarrow$ CubicalDbl(E, nP)
6: **return** (nP, nPQ)

Algorithm 8.17 CubicalTranslate$(\widetilde{P}, \widetilde{T})$

Input: Cubical Montgomery coordinates $\widetilde{P} = (X(P), Z(P))$ and $\widetilde{T} = (X(T), Z(T))$ where $T = (X(T) : Z(T))$ is a 2-torsion point.
Output: The cubical translation $\widetilde{P + T} = (X(P + T), Z(P + T))$.

1: $X \leftarrow X(T)X(P) - Z(T)Z(P)$
2: $Z \leftarrow Z(T)X(P) - X(T)Z(P)$
3: **if** $Z(T) = 0$ **then**
4: $\quad Z \leftarrow -Z$
5: **if** $X(T) = 0$ **then**
6: $\quad X \leftarrow -X$
7: **return** (X, Z)

Algorithm 8.18 CubicalRatio$(\widetilde{P_1}, \widetilde{P_2})$

Input: Cubical Montgomery coordinates $\widetilde{P_1} = (X(P_1), Z(P_1))$ and $\widetilde{P_2} = (X(P_2), Z(P_2))$ such that $P_1 = (X(P_1) : Z(P_1)) = P_2 = (X(P_2) : Z(P_2))$.
Output: The ratio λ such that $X(P_2) = \lambda X(P_1)$ and $Z(P_2) = \lambda Z(P_1)$.

1: **if** $X(P_1) = 0$ **then**
2: **return** $Z(P_2)/Z(P_1)$
3: **else**
4: **return** $X(P_2)/X(P_1)$

Algorithm 8.19 Tate$(E, e, x(P), x(Q), x(P+Q))$

Input: A supersingular elliptic curve $E : y^2 = x^3 + Ax^2 + x/\mathbb{F}_{p^2}$ in Montgomery model, an integer e, the x-coordinates of the points $P, Q, P+Q \in E(\mathbb{F}_{p^2})$ where $2^e P = 0_E$.
Output: The reduced Tate pairing $t_{2^e}(P, Q) \in \mu_{2^e}$.

1: $nP, nPQ \leftarrow$ CubicalLadder$(E, e-1, (x(P+Q), 1), (x(P), 1), x(Q))$
2: $O \leftarrow$ CubicalTranslate(nP, nP)
3: $Q' \leftarrow$ CubicalTranslate(nPQ, nP)
4: $\lambda \leftarrow$ CubicalRatio$(P, P')/$CubicalRatio$((1, 0), O)$
5: **return** $\lambda^{(p^2-1)/2^e}$ // Using Eq. (10)

8.4. Isogenies

Below we give pseudocode for the x-only elliptic curve isogenies that our implementation uses. SQIsign only requires isogenies of degree 2^n, which we compute through chains of 2- resp. 4-isogenies. These individual isogenies use the following algorithms:

- TwoIsogenyCodomain (Algorithm 8.20) takes as input the projective x-coordinate of a kernel generator $P \in E$ of order 2 with $P \neq (0 : 1)$, and outputs the codomain curve E' of the 2-isogeny of kernel $\langle P \rangle$ represented by the projective Montgomery coefficient $(A'_{24} : C'_{24})$.
- TwoIsogenyEval (Algorithm 8.21) takes as input the projective x-coordinates of a kernel generator $P \in E$ of order 2 with $P \neq (0 : 1)$ and of a point $Q \in E$, and outputs the projective x-coordinate of the image $Q' \in E'$ of Q under the isogeny computed by TwoIsogenyCodomain.
- TwoIsogenyCodomainSingular (Algorithm 8.22) and TwoIsogenyEvalSingular (Algorithm 8.23) are equivalent to TwoIsogenyCodomain and TwoIsogenyEval for the case that $P = (0 : 1)$.
- FourIsogenyCodomain (Algorithm 8.24) takes as input the projective x-coordinate of a kernel generator $P \in E$ of order 4 with $[2]P \neq (0 : 1)$, and outputs the codomain curve E' of the 4-isogeny of kernel $\langle P \rangle$ represented by the projective Montgomery coefficient $(A'_{24} : C'_{24})$ and constants `consts` needed for FourIsogenyEval.
- FourIsogenyEval (Algorithm 8.25) takes as input the projective x-coordinate of a point $Q \in E$, and the constants `consts` returned by FourIsogenyCodomain. It outputs the projective x-coordinate of the image $Q' \in E'$ of Q under the isogeny computed by FourIsogenyCodomain.

Using these basic operations, we can describe the following computation of a 2^e-isogeny: TwoIsogenyChain (Algorithm 8.26) takes as input the projective x-coordinate of a kernel generator $P \in E$ of order 2^e, the Montgomery coefficient $(A'_{24} : C'_{24})$ of E, the positive integer e, and a list of points `pts` to be evaluated. It outputs the codomain curve E' of the 2^e-isogeny of kernel $\langle P \rangle$ represented by the projective Montgomery coefficient $(A'_{24} : C'_{24})$, and the list of evaluated points `pts'`. The algorithm computes a chain of $\lfloor e/2 \rfloor$ isogenies of degree 4, followed by a final 2-isogeny if e is odd, using balanced strategies.

When computing 2^e-isogenies from a given point $P \in E$ of order 2^e, we must verify the correctness of the order of P within the computation of the chain of isogenies. We do so by verifying we have a point of order 4 (if $e > 1$) or a point of order 2 (if $e = 1$) in the first isogeny step of the computation of this chain. Furthermore, for honestly generated inputs, our implementation choices guarantee that the special cases of $P = (0 : 1)$ resp. $[2]P = (0 : 1)$ do not occur in TwoIsogenyChain. Therefore, verification rejects signatures that lead to these cases.

We further compute small chains of 2-isogenies that can include the special case of a kernel generator $P = (0 : 1)$ only during signing. Due to the small length e, we use a naive implementation without strategies, as shown in TwoIsogenyChainSmall (Algorithm 8.27).

Algorithm 8.20 TwoIsogenyCodomain(P)

Input: Projective point $P = (X_P : Z_P) \in E$ of order 2 with $P \neq (0 : 1)$.
Output: The coefficient $(A'_{24} : C'_{24})$ corresponding to the image curve $E' := E/\langle P \rangle$ under the 2-isogeny.

$A'_{24} \leftarrow X_P^2$
$C'_{24} \leftarrow Z_P^2$
$A'_{24} \leftarrow C'_{24} - A'_{24}$
return $(A'_{24} : C'_{24})$

Algorithm 8.21 TwoIsogenyEval(P, Q)

Input: Projective point $P = (X_P : Z_P)$ of order 2 with $P \neq (0 : 1)$, and projective point $Q = (X_Q : Z_Q)$ where $P, Q \in E$.
Output: Projective point $Q' = (X_{Q'} : Z_{Q'}) \in E'$ corresponding to the image of Q under the 2-isogeny defined by P.

$t_0 \leftarrow X_P + Z_P$
$t_1 \leftarrow X_P - Z_P$
$t_2 \leftarrow X_Q + Z_Q$
$t_3 \leftarrow X_Q - Z_Q$
$t_0 \leftarrow t_0 \cdot t_3$
$t_1 \leftarrow t_1 \cdot t_2$
$t_2 \leftarrow t_0 + t_1$
$t_3 \leftarrow t_0 - t_1$
$X_{Q'} \leftarrow X_Q \cdot t_2$
$Z_{Q'} \leftarrow Z_Q \cdot t_3$
return $(X_{Q'} : Z_{Q'})$

Algorithm 8.22 TwoIsogenyCodomainSingular$((A_{24} : C_{24}))$

Input: Montgomery coefficient $(A_{24} : C_{24})$ of the curve E.
Output: The coefficients $(A'_{24} : C'_{24})$ corresponding to the image curve $E' := E/\langle P \rangle$ under the 2-isogeny with $P = (0 : 1)$, and constants `consts` needed for TwoIsogenyEvalSingular.

$t_0 \leftarrow A_{24} + A_{24}$
$t_0 \leftarrow t_0 - C_{24}$
$t_0 \leftarrow t_0 + t_0$
$t_1 \leftarrow C_{24}^{-1}$
$t_0 \leftarrow t_0 \cdot t_1$
$c_1 \leftarrow t_0$
$A'_{24} \leftarrow t_0 + t_0$
$t_0 \leftarrow t_0^2$
$t_0 \leftarrow t_0 - 4$
$t_0 \leftarrow \sqrt{t_0}$
$c_1 \leftarrow -t_0$
$C'_{24} \leftarrow t_0 + t_0$
$A'_{24} \leftarrow A'_{24} + C'_{24}$
$C'_{24} \leftarrow C'_{24} + C'_{24}$
`consts` $\leftarrow (c_0, c_1)$
return $(A'_{24} : C'_{24})$, `consts`

Algorithm 8.23 TwoIsogenyEvalSingular$(Q, \texttt{consts})$

Input: Projective point $Q = (X_Q : Z_Q)$ where $Q \in E$, constants `consts` returned by TwoIsogenyCodomainSingular.
Output: Projective point $Q' = (X_{Q'} : Z_{Q'}) \in E'$ corresponding to the image of Q under the 2-isogeny defined by $P = (0 : 1)$.

$c_0, c_1 \leftarrow \texttt{consts}$
$t_0 \leftarrow X_Q \cdot Z_Q$
$t_1 \leftarrow c_0 \cdot Z_Q$
$t_1 \leftarrow t_1 + X_Q$
$t_1 \leftarrow t_1 \cdot X_Q$
$X_{Q'} \leftarrow Z_Q^2$
$X_{Q'} \leftarrow X_{Q'} + t_1$
$Z_{Q'} \leftarrow t_0 \cdot c_1$
return $(X_{Q'} : Z_{Q'})$

Algorithm 8.24 FourIsogenyCodomain(P)

Input: Projective point $P = (X_P : Z_P) \in E$ of order 4 such that $[2]P \neq (0 : 1)$.
Output: The coefficient $(A'_{24} : C'_{24})$ corresponding to the image curve $E' := E/\langle P \rangle$ under the 4-isogeny, and constants `consts` needed for FourIsogenyEval.

$c_0 \leftarrow Z_P^2$
$c_1 \leftarrow X_P - Z_P$
$c_2 \leftarrow X_P + Z_P$
$t_0 \leftarrow X_P^2$
$t_3 \leftarrow c_0 + t_0$
$t_4 \leftarrow c_0 - t_0$
$A'_{24} \leftarrow t_3 \cdot t_4$
$C'_{24} \leftarrow c_0^2$
$c_0 \leftarrow c_0 + c_0$
$c_0 \leftarrow c_0 + c_0$
$\texttt{consts} \leftarrow (c_0, c_1, c_2)$
return $(A'_{24} : C'_{24}), \texttt{consts}$

Algorithm 8.25 FourIsogenyEval$(Q, \texttt{consts})$

Input: Projective point $Q = (X_Q : Z_Q)$ where $Q \in E$, and constants `consts` returned by FourIsogenyCodomain.
Output: Projective point $Q' = (X_{Q'} : Z_{Q'}) \in E'$ corresponding to the image of Q under the 4-isogeny defined by P.

$c_0, c_1, c_2 \leftarrow \texttt{consts}$
$t_0 \leftarrow X_Q + Z_Q$
$t_1 \leftarrow X_Q - Z_Q$
$X_{Q'} \leftarrow t_0 \cdot c_1$
$Z_{Q'} \leftarrow t_1 \cdot c_2$
$t_0 \leftarrow t_0 \cdot t_1$
$t_0 \leftarrow t_0 \cdot c_0$
$t_1 \leftarrow X_{Q'} + Z_{Q'}$
$Z_{Q'} \leftarrow X_{Q'} - Z_{Q'}$
$t_1 \leftarrow t_1^2$
$Z_{Q'} \leftarrow Z_{Q'}^2$
$X_{Q'} \leftarrow t_0 + t_1$
$t_0 \leftarrow t_0 - Z_{Q'}$
$X_{Q'} \leftarrow X_{Q'} \cdot t_1$
$Z_{Q'} \leftarrow Z_{Q'} \cdot t_0$
return $(X_{Q'} : Z_{Q'})$

Algorithm 8.26 TwoIsogenyChain$(P, (A_{24} : C_{24}), e, \texttt{pts})$

Input: Projective point $P = (X_P : Z_P) \in E$ of order 2^e, Montgomery coefficient $(A_{24} : C_{24})$ of the curve E, a positive integer e, a list of points pts to be evaluated.

Output: The coefficient $(A'_{24} : C'_{24})$ corresponding to the image curve $E' := E/\langle P \rangle$ under the 2^e-isogeny, list of evaluated points pts$'$.

$(A'_{24} : C'_{24}) \leftarrow (A_{24} : C_{24})$
Initialize a list strat_pts $\leftarrow [P]$
Initialize a list orders $\leftarrow [e]$
$k \leftarrow 0$
for j **from** 0 **up to** $\lfloor e/2 \rfloor - 1$ **do** // Chain of 4-isogenies
  while orders$[k] \neq 2$ **do**
    $k \leftarrow k + 1$
    $n \leftarrow \lfloor \texttt{orders}[k-1]/4 \rfloor \cdot 2 + (\texttt{orders}[k-1] \pmod 2)$
    orders$[k] \leftarrow$ orders$[k-1] - n$
    strat_pts$[k] \leftarrow$ strat_pts$[k-1]$
    for i **from** 0 **up to** $n - 1$ **do**
      strat_pts$[k] \leftarrow$ xDBL(strat_pts$[k], (A_{24} : C_{24})$)
  if $j = 0$ **then** // Input point validation
    $K \leftarrow$ strat_pts$[k]$
    if $[2]K = 0_E$ **or** $[4]K \neq 0_E$ **then**
      raise Exception: ("TwoIsogenyChain failed: wrong point order")
    if $[2]K = (0 : 1)$ **then**
      raise Exception: ("TwoIsogenyChain failed: unexpected singular 4-isogeny")
  $((A'_{24} : C'_{24}), \texttt{consts}) \leftarrow$ FourIsogenyCodomain(K)
  strat_pts $\leftarrow$ [FourIsogenyEval$(Q, \texttt{consts}) \mid Q \in$ strat_pts]
  orders $\leftarrow [m - 2 \mid m \in$ orders]
  $k \leftarrow k - 1$
  pts $\leftarrow$ [FourIsogenyEval$(Q, \texttt{consts}) \mid Q \in$ pts]
if $e \pmod 2 = 1$ **then** // Final 2-isogeny if e is odd
  $K \leftarrow$ strat_pts$[0]$
  if $e = 1$ **then** // Input point validation.
    if $K = 0_E$ **or** $[2]K \neq 0_E$ **then**
      raise Exception: ("TwoIsogenyChain failed: wrong point order")
    if $K = (0 : 1)$ **then**
      raise Exception: ("TwoIsogenyChain failed: unexpected singular 2-isogeny")
  $(A'_{24} : C'_{24}) \leftarrow$ TwoIsogenyCodomain(K)
  pts $\leftarrow$ [TwoIsogenyEval$(Q, K) \mid Q \in$ pts]
return $(A'_{24} : C'_{24})$, pts

Algorithm 8.27 TwoIsogenyChainSmall$(P, (A_{24} : C_{24}), e, \texttt{pts}, \texttt{cond})$

Input: Projective point $P = (X_P : Z_P) \in E$ of order 2^e, Montgomery coefficient $(A_{24} : C_{24})$ of the curve E, a positive integer e, a list of points pts to be evaluated, a boolean cond signifying if the function is called during signing or verifying.

Output: The coefficient $(A'_{24} : C'_{24})$ corresponding to the image curve $E' := E/\langle P \rangle$ under the 2^e-isogeny, list of evaluated points pts$'$

$(A'_{24} : C'_{24}) \leftarrow (A_{24} : C_{24})$
for i **from** 0 **up to** $e - 1$ **do**
  $K \leftarrow P$
  for j **from** 0 **up to** $e - i - 2$ **do**
    $K \leftarrow$ xDBL$(P, (A_{24} : C_{24}))$
  if $i = 0$ **then**
    if $[2]K \neq 0_E$ **then**
      raise Exception: ("TwoIsogenyChainSmall failed: wrong order") // Can only occur during verification
    if $K = (0 : 1)$ **then**
      if cond $=$ true **then** // Function called during signing
        $((A'_{24} : C'_{24}), \texttt{consts}) \leftarrow$ TwoIsogenyCodomainSingular$((A'_{24} : C'_{24}))$
        $P \leftarrow$ TwoIsogenyEvalSingular$(P, \texttt{consts})$
        pts $\leftarrow$ [TwoIsogenyEvalSingular$(Q, \texttt{consts}) \mid Q \in \texttt{pts}$]
      else // Function called during verification.
        raise Exception: ("TwoIsogenyChainSmall failed: unexpected singular 2-isogeny")
  else
    $(A'_{24} : C'_{24}) \leftarrow$ TwoIsogenyCodomain(K)
    $P \leftarrow$ TwoIsogenyEval(K, P)
    pts $\leftarrow$ [TwoIsogenyEval$(K, Q) \mid Q \in \texttt{pts}$]
return $(A'_{24} : C'_{24}), \texttt{pts}$

8.5. Theta coordinates

We now depict how two-dimensional isogenies of degree 2 (i.e., $(2, 2)$-isogenies) are implemented using theta coordinates of level 2. This implementation is built upon the mathematical insights from [CDLR25; DMPR24]. For each algorithm, we give its cost in terms of squarings, multiplications, inversions, and additions over the base field $\mathbb{F}_{p^2}$, denoted $\mathbf{S}$, $\mathbf{M}$, $\mathbf{I}$, and $\mathbf{a}$, respectively.

8.5.1. Theta coordinates of level 2

For efficiency reasons, we do not use the Jacobian model for arithmetic operations on PPAS and instead use *theta coordinates*. This coordinate system can be seen as a generalization of x-only arithmetic, where the x-coordinate is used for efficient arithmetic on Montgomery curves. In this section, we detail how theta coordinates are defined on Montgomery curves and on PPAS.

8.5.1.1. On Montgomery curves. Let E be a Montgomery curve defined over $\mathbb{F}_q$ by the equation $By^2 = x^3 + Ax^2 + x$ with $A, B \in \mathbb{F}_q$. As mentioned in Section 8.2.1, we use x-only or $(X : Z)$-only arithmetic. *Theta coordinates of level 2*, denoted by $(\theta_0 : \theta_1)$, are alternative projective coordinates to $(X : Z)$-coordinates. They also determine points up to sign. A system of theta coordinates (of level 2) on E is fully determined by the *theta null point* $(a : b)$ whose coordinates are called the *theta constants*.

The theta null point $(a : b)$ is determined by a basis (T'_1, T'_2) of $E[4]$, where $T'_1 = (-1 : 1)$ and $T'_2 = (r : s)$ in $(X : Z)$-coordinates: $(a : b) = (r + s : r - s)$. Given the theta null point $(a : b)$, we obtain change of coordinates formulas between Montgomery $(X : Z)$-coordinates and theta coordinates:

$$\begin{aligned} (X : Z) &\longrightarrow (\theta_0 : \theta_1) = (a(X - Z) : b(X + Z)) \\ (X : Z) = (a\theta_1 + b\theta_0 : a\theta_1 - b\theta_0) &\longleftarrow (\theta_0 : \theta_1), \end{aligned}$$

using the convention that $0_E = (1 : 0)$ in $(X : Z)$-coordinates. Hence, such a basis (T'_1, T'_2) of $E[4]$ is sufficient to determine a system of theta coordinates (of level 2) on E, and different bases may generate different systems

of theta coordinates. Assuming we have computed the theta null point $(a : b)$, we depict the change of coordinate procedure in Algorithm 8.28.

Algorithm 8.28 MontgomeryToTheta$(P, 0)$

Input: A point $P := (X : Z)$ in Montgomery coordinates, and theta null point $0 := (a : b)$.
Output: Point P in theta coordinates $(\theta_0 : \theta_1)$.

1: $\theta_0 \leftarrow X - Z$
2: $\theta_0 \leftarrow a \cdot \theta_0$
3: $\theta_1 \leftarrow X + Z$
4: $\theta_1 \leftarrow b \cdot \theta_1$
5: **return** $(\theta_0 : \theta_1)$ // Total cost: $2\mathbf{M} + 2\mathbf{a}$

8.5.1.2. On principally polarized abelian surfaces. Let A be a PPAS defined over $\mathbb{F}_q$. A system of theta coordinates (of level 2) on A consists of projective coordinates $(x : y : z : w)$ (possibly defined over an extension of $\mathbb{F}_q$). Similarly to the elliptic-curve case, if A is a Jacobian, the coordinates $(x : y : z : w)$ determine points up to sign. If $A = E_1 \times E_2$ is a product of elliptic curves, then $(x : y : z : w)$ determines a pair of elliptic points $(\pm P, \pm Q)$ up to sign, component-wise, and thus represents up to four points of the product surface.

As before, a system of theta coordinates (of level 2) on the PPAS A, along with A itself, is fully determined by the theta null point $(a : b : c : d) := (x(0) : y(0) : z(0) : w(0))$. There exist different systems of theta coordinates on the same PPAS and appropriate change of coordinates formulas [Dar24, Theorem 12]. In the algorithms that follow, we fix a choice of theta coordinate system, following [DMPR24].

8.5.1.3. Product theta coordinates. Let $A := E_1 \times E_2$ be a product of (Montgomery) elliptic curves, each with their own system of theta coordinates (of level 2), say $(\theta_0 : \theta_1)$ and $(\theta'_0 : \theta'_1)$ respectively, as defined in Section 8.5.1.1. We define the *product theta coordinates* on A by:

$$(x : y : z : w) := (\theta_0\theta'_0 : \theta_1\theta'_0 : \theta_0\theta'_1 : \theta_1\theta'_1).$$

Not all systems of theta coordinates on A are product theta coordinates. When computing a gluing isogeny $\Phi_1 : E_1 \times E_2 \longrightarrow A_1$, we may have to change coordinates from a product to a non-product system of theta coordinates (see Algorithm 8.38). Similarly, when we compute a splitting isogeny $\Phi_e : A_{e-1} \longrightarrow E_3 \times E_4$, the system of theta coordinates on the codomain $E_3 \times E_4$ may not be of product form, and so we may have to compute a change of coordinates to express image points component-wise in $E_3 \times E_4$ (see Algorithm 8.43).

8.5.2. Doubling formulas using theta coordinates

We consider a PPAS A with a system of theta coordinates of level 2 induced by the theta null point $(a : b : c : d)$. We introduce the *Hadamard* and the *squaring* operators acting on theta-coordinates as follows:

$$\mathcal{H}(x, y, z, w) := (x + y + z + w, x - y + z - w, x + y - z - w, x - y - z + w),$$
$$\mathcal{S}(x, y, z, w) := (x^2, y^2, z^2, w^2).$$

which cost $8\mathbf{a}$ and $4\mathbf{S}$, respectively (using recursive formulae for the Hadamard operator). The Hadamard operator maps theta coordinates to another system of theta coordinates called *dual theta coordinates*. This system of theta coordinates is determined by the *dual theta null point* $(a' : b' : c' : d')$ with $a', b', c', d' := \mathcal{H}(a, b, c, d)$. A level 2 theta null point encodes a canonical 2-isogeny $A \to B$. We will call the dual of the theta null point of B the dual isogeneous theta null point, and denote its coordinates by $(\alpha : \beta : \gamma : \delta)$. We have $(\alpha^2 : \beta^2 : \gamma^2 : \delta^2) = \mathcal{H}(a^2 : b^2 : c^2 : d^2)$.

Given a point $P = (x : y : z : w)$ expressed in theta coordinates, ThetaDBL given in Algorithm 8.30 describes how to compute $[2]P$. ThetaDBL requires auxiliary constants as input, which depend only on the theta null point. The precomputation of these constants is done by the algorithm ThetaPrecomp, as described in Algorithm 8.29.

For the rest of this section, for clarity of exposition we will denote the application of ThetaDBL to a point P as $[2]P$ (where one should assume that the constants `consts` needed are accessible, e.g., by applying ThetaPrecomp or as an output of the isogeny algorithms).

Algorithm 8.29 ThetaPrecomp(0_A)

Input: Theta null point $0_A := (a : b : c : d)$, and optionally the dual isogeneous theta null point $0'_A := (\alpha : \beta : \gamma : \delta)$.
Output: Auxiliary constants $\texttt{consts} := \{abc, abd, acd, bcd, \alpha^2\beta^2\gamma^2, \alpha^2\beta^2\delta^2, \alpha^2\gamma^2\delta^2, \beta^2\gamma^2\delta^2\}$.

1: **if** $0'_A$ **then**
2: $(\alpha^2, \beta^2, \gamma^2, \delta^2) \leftarrow \mathcal{S}(\alpha, \beta, \gamma, \delta)$
3: **else**
4: $(\alpha^2, \beta^2, \gamma^2, \delta^2) \leftarrow \mathcal{H} \circ \mathcal{S}(a, b, c, d)$
5: $t_1 \leftarrow \alpha^2 \cdot \beta^2$
6: $t_2 \leftarrow \gamma^2 \cdot \delta^2$
7: $\alpha^2\beta^2\gamma^2 \leftarrow t_1 \cdot \gamma^2$
8: $\alpha^2\beta^2\delta^2 \leftarrow t_1 \cdot \delta^2$
9: $\alpha^2\gamma^2\delta^2 \leftarrow t_2 \cdot \alpha^2$
10: $\beta^2\gamma^2\delta^2 \leftarrow t_2 \cdot \beta^2$
11: $t_1 \leftarrow a \cdot b$
12: $t_2 \leftarrow c \cdot d$
13: $abc \leftarrow t_1 \cdot c$
14: $abd \leftarrow t_1 \cdot d$
15: $acd \leftarrow t_2 \cdot a$
16: $bcd \leftarrow t_2 \cdot b$
17: $\texttt{consts} \leftarrow \{abc, abd, acd, bcd, \alpha^2\beta^2\gamma^2, \alpha^2\beta^2\delta^2, \alpha^2\gamma^2\delta^2, \beta^2\gamma^2\delta^2\}$
18: **return** consts // Total cost: $4\mathbf{S} + 12\mathbf{M}(+8\mathbf{a})$

Algorithm 8.30 ThetaDBL$(P, \texttt{consts})$

Input: The theta coordinates of P on A with theta null point $0_A := (a : b : c : d)$, and the auxiliary constants $\texttt{consts} :=$ ThetaPrecomp(0_A).
Output: The theta coordinates of the point $[2]P$.

1: $x_P, y_P, z_P, w_P \leftarrow P$
2: $c_1, c_2, c_3, c_4, c_5, c_6, c_7, c_8 \leftarrow \texttt{consts}$
3: $X_{2P}, Y_{2P}, Z_{2P}, W_{2P} \leftarrow \mathcal{S} \circ \mathcal{H} \circ \mathcal{S}(x_P, y_P, z_P, w_P)$
4: $X_{2P} \leftarrow X_{2P} \cdot c_8$
5: $Y_{2P} \leftarrow Y_{2P} \cdot c_7$
6: $Z_{2P} \leftarrow Z_{2P} \cdot c_6$
7: $W_{2P} \leftarrow W_{2P} \cdot c_5$
8: $X_{2P}, Y_{2P}, Z_{2P}, W_{2P} \leftarrow \mathcal{H}(X_{2P}, Y_{2P}, Z_{2P}, W_{2P})$
9: $X_{2P} \leftarrow X_{2P} \cdot c_4$
10: $Y_{2P} \leftarrow Y_{2P} \cdot c_3$
11: $Z_{2P} \leftarrow Z_{2P} \cdot c_2$
12: $W_{2P} \leftarrow W_{2P} \cdot c_1$
13: **return** $(X_{2P} : Y_{2P} : Z_{2P} : W_{2P})$ // Total cost: $8\mathbf{S} + 8\mathbf{M} + 16\mathbf{a}$

8.5.3. Generic $(2, 2)$-isogeny computation

We first explain how to compute a generic $(2, 2)$-isogeny $\Phi : A \longrightarrow B$ given torsion points lying above $\ker(\Phi)$. We say that two points $T_1, T_2 \in A[2^k]$, for some $k > 1$, lie above the kernel of a $(2, 2)$-isogeny Φ if $\ker(\Phi) = \langle [2^{k-1}]T_1 \rangle \oplus \langle [2^{k-1}]T_2 \rangle$. We explain in Section 2.4.1 why we can assume such points are accessible. We assume that $\ker(\Phi)$ satisfies a compatibility condition with the system of theta coordinates on A for the following formulas to work. When working with a chain of $(2, 2)$-isogenies, it will be sufficient to ensure this compatibility condition is satisfied at the start of the chain, as this will propagate along the chain. The reader may refer to [DMPR24, Sections 2.3 and 4] for more details on this.

8.5.3.1. With isotropic 8-torsion lying above the kernel. Assume we know the theta coordinates of two points $T'_1, T'_2 \in A[8]$ of order 8 that are isotropic, namely $e_8(T'_1, T'_2) = 1$, and such that $\ker(\Phi) = \langle [4]T'_1 \rangle \oplus \langle [4]T'_2 \rangle$. Using the algorithm GenericCodomainWith8Torsion (given in Algorithm 8.31) we can compute the theta null

point $(a_2 : b_2 : c_2 : d_2)$ of the codomain B along with its dual $(\alpha : \beta : \gamma : \delta)$ and the projective inverse $(\alpha^{-1} : \beta^{-1} : \gamma^{-1} : \delta^{-1})$. Note that in the general case, we expect $\alpha\beta\gamma\delta \neq 0$, indeed otherwise A would correspond to a product of elliptic curves. As such, we assume this in GenericCodomainWith8Torsion. This data is enough to determine the isogeny Φ. In particular, $(\alpha^{-1} : \beta^{-1} : \gamma^{-1} : \delta^{-1})$ can be used to evaluate Φ on points (see Algorithm 8.35). The theta null point of B and its dual are not used for the evaluation but are useful for the doubling formulas.

There are two failures that can occur during the computation of a chain of $(2,2)$-isogenies with repeated use of the algorithm GenericCodomainWith8Torsion.

The codomain of all $(2,2)$-isogenies except for the last is a product of elliptic curves (i.e., the isogeny chain splits in the middle). We must check that, at each step (except the last), the coordinates of the dual isogeneous theta null point $(\alpha : \beta : \gamma : \delta)$ are non-zero (see Line 5 in Algorithm 8.31). Indeed, if this is the case we are doing a gluing isogeny, but since we checked explicitely that our first isogeny was a gluing, this means we hit a random splitting earlier in the chain.

At each step, the input 4-torsion points $[2]T_1'', [2]T_2''$ are not isotropic nor compatible with the theta structure. In practice, we do this by checking that the image of the 8-torsion points under the isogeny Φ (namely P, Q defined in Lines 18 and 19) are of the form $(x : x : y : y)$ and $(z : w : z : w)$, respectively. This ensures that:

- They are of 4-torsion;
- They are above the correct 2-torsion points on the codomain (the kernel of the next isogeny in the chain);
- They are compatible with the theta structure on the codomain.

Note that this ensures the kernel of the next isogeny is isotropic and compatible with the theta structure, and so we do not need the check at the last isogeny of the chain. We depict this check in the algorithm CheckIsotropic.

To bootstrap these checks, for our very first gluing kernel we do the following check on the basis $[4]T_1'' = (P_1, Q_1), [4]T_2'' = (P_2, Q_2)$ (see Line 1 of Algorithm 8.37). We check that all points P_1, Q_1, P_2, Q_2 are not 0, and that $P_1 \neq Q_1, P_2 \neq Q_2$, this ensures that our kernel is not diagonal and isotropic. We also note that Algorithm 8.37 ensures that our points of 4-torsion $[2]T_1'', [2]T_2''$ are compatible with our initial theta structure.

As discussed in Chapter 9, heuristically the probability of the first failure occurring is $O(1/p)$, and therefore we choose to forgo performing these checks during signing. We remark that, due to this, the checks are not required in GenericCodomainWith4Torsion or GenericCodomain, as these are only run during signing.

Also, we can ignore the isotropy checks for signing, because we control the kernels in the signing procedure. However, they must be performed during verification to ensure that the signature has not been malformed.

8.5.3.2. With isotropic 4-torsion lying above the kernel. When we cannot access 8-torsion points lying above $\ker(\Phi)$ but only 4-torsion points T_1', T_2' such that $\ker(\Phi) = \langle [2]T_1', [2]T_2' \rangle$, we can still compute Φ using the algorithm GenericCodomainWith4Torsion given by Algorithm 8.33. It only requires T_1' and the theta null point of A. However, it is much more costly than GenericCodomainWith8Torsion as it involves two square roots.

8.5.3.3. With kernel generators only. When we only know generators T_1, T_2 of $\ker(\Phi)$, we can compute Φ with GenericCodomain given the theta null point of A only at the expense of three square roots. Interestingly, due to the compatibility condition of $\ker(\Phi)$ with our system of theta coordinates, the points T_1 and T_2 are not needed for the computation. We refer to Algorithm 8.34, where again we assume $\alpha\beta\gamma\delta \neq 0$.

8.5.4. Generic $(2,2)$-isogeny evaluation

Assume we have computed the codomain of a $(2,2)$-isogeny $\Phi : A \longrightarrow B$ with GenericCodomainWith8Torsion, GenericCodomainWith4Torsion or GenericCodomain. Namely, we have the theta null point 0_B of B, its dual $(\alpha : \beta : \gamma : \delta)$ and the dual projective inverse $(\alpha^{-1} : \beta^{-1} : \gamma^{-1} : \delta^{-1})$. We are now able to evaluate Φ at a point $P \in A$ using only the theta coordinates of P and $(\alpha^{-1} : \beta^{-1} : \gamma^{-1} : \delta^{-1})$ with GenericEval. We describe this in Algorithm 8.35.

8.5.5. Gluing $(2,2)$-isogeny computation

We now focus on a gluing $(2,2)$-isogeny $\Phi : E_1 \times E_2 \to A$, where E_1, E_2 are elliptic curves and A is a PPAS.

Algorithm 8.31 GenericCodomainWith8Torsion(T_1'', T_2'')

Input: Theta coordinates of T_1'' and T_2'', such that $\ker(\Phi) = \langle [4]T_1'' \rangle \oplus \langle [4]T_2'' \rangle$.
Output: Dual isogeneous theta null point $(\alpha : \beta : \gamma : \delta)$, the inverse of the dual isogeneous theta null point $(\alpha^{-1} : \beta^{-1} : \gamma^{-1} : \delta^{-1})$, and the theta null point 0_B on B. // Case $\alpha \cdot \beta \cdot \gamma \cdot \delta \neq 0$.

1: $x_{T_1''}, y_{T_1''}, z_{T_1''}, w_{T_1''} \leftarrow T_1''$
2: $x_{T_2''}, y_{T_2''}, z_{T_2''}, w_{T_2''} \leftarrow T_2''$
3: $(x\alpha, x\beta, y\gamma, y\delta) \leftarrow \mathcal{H} \circ \mathcal{S}(x_{T_1''}, y_{T_1''}, z_{T_1''}, w_{T_1''})$
4: $(z\alpha, w\beta, z\gamma, w\delta) \leftarrow \mathcal{H} \circ \mathcal{S}(x_{T_2''}, y_{T_2''}, z_{T_2''}, w_{T_2''})$
5: **if** $0 \in \{x\alpha, x\beta, z\alpha, w\beta, z\gamma, w\delta\}$ **then**
6: **raise** Exception: ("GenericCodomainWith8Torsion failed: unexpected splitting") // Check only done in verification.
7: $x\alpha w\beta \leftarrow x\alpha \cdot w\beta$
8: $z\alpha x\beta \leftarrow z\alpha \cdot x\beta$
9: $\alpha \leftarrow z\alpha \cdot x\alpha w\beta$
10: $\beta \leftarrow w\beta \cdot z\alpha x\beta$
11: $\gamma \leftarrow z\gamma \cdot x\alpha w\beta$
12: $\delta \leftarrow w\delta \cdot z\alpha x\beta$
13: $z\gamma w\delta \leftarrow z\gamma \cdot w\delta$
14: $\alpha^{-1} \leftarrow x\beta \cdot z\gamma w\delta$
15: $\beta^{-1} \leftarrow x\alpha \cdot z\gamma w\delta$
16: $\gamma^{-1} \leftarrow \delta$
17: $\delta^{-1} \leftarrow \gamma$
18: $P \leftarrow (x\alpha : x\beta : y\gamma : y\delta)$
19: $Q \leftarrow (z\alpha : w\beta : z\gamma : w\delta)$
20: $I \leftarrow (\alpha^{-1} : \beta^{-1} : \gamma^{-1} : \delta^{-1})$
21: **if not** CheckIsotropic(P, Q, I) **then**
22: **raise** Exception: ("GenericCodomainWith8Torsion failed: P, Q not isotropic") // Check only done in verification.
23: $(a_2, b_2, c_2, d_2) \leftarrow \mathcal{H}(\alpha, \beta, \gamma, \delta)$
24: $0_B \leftarrow (a_2 : b_2 : c_2 : d_2)$
25: **return** $(\alpha : \beta : \gamma : \delta), (\alpha^{-1} : \beta^{-1} : \gamma^{-1} : \delta^{-1}), 0_B$ // Total cost (without checks): $8\mathbf{S} + 9\mathbf{M} + 24\mathbf{a}$

The first step is to apply a change of basis matrix to move points $P = (P_1, P_2)$ on $E_1 \times E_2$, where each P_i is in theta coordinates (by applying MontgomeryToTheta) to (non-product) theta coordinates corresponding to our choice of theta structure, as discussed briefly in Section 8.5.1.3. This is given in ProductToTheta.

In GluingCodomain (see Algorithm 8.39) we present the algorithm for computing the codomain A of the gluing $(2,2)$-isogeny $\Phi : E_1 \times E_2 \to A$. Explicitly, due to the way we set up our theta structure in Algorithm 8.37, we will always have $\delta = 0$ (otherwise, gluing has failed, for instance because A is a product, and we abort). On input 8-torsion points T_1'', T_2'' lying above the kernel generators, GluingCodomain outputs the dual isogeneous theta point $(\alpha : \beta : \gamma : 0)$, the "inverse" of the dual isogeneous theta null point $(\alpha^{-1} : \beta^{-1} : \gamma^{-1} : 0)$, and the theta null point $(a_2 : b_2 : c_2 : d_2)$ on A.

As in the generic case, there are two failures that can occur during gluing. First, the algorithm we present assumes that for points on the dual surface (i.e., after applying $\mathcal{H} \circ \mathcal{S}$) the final coordinate is zero. This check is performed in Line 9 of GluingCodomain. On the other hand, we must check that the other projective factors of such points are non-zero: see Line 11. Furthermore, analogously to the generic case, we need to ensure that the input points T_1'', T_2'' are isotropic and compatiblew with the theta structure. This is handled in Line 22 of GluingCodomain.

8.5.6. Gluing $(2,2)$-isogeny evaluation

Assume that we have computed the codomain A of a gluing $(2,2)$-isogeny $\Phi : E_1 \times E_2 \longrightarrow A$ with GluingCodomain (again, due to the theta structure set-up in Algorithm 8.37, we have that $\delta = 0$). Namely, we have the theta null point $(a_2 : b_2 : c_2 : d_2)$ of A, its dual $(\alpha : \beta : \gamma : 0)$, the dual projective "inverse" $(\alpha^{-1}, \beta^{-1} : \gamma^{-1} : 0)$ and $(x : x : y : y)$ the dual of theta point $\Phi(T_1'')$ on A. We are now able to evaluate Φ at a point $P \in E_1 \times E_2$ using the GluingEval

Algorithm 8.32 CheckIsotropic(P, Q, I)

Input: Theta coordinates of $P := \mathcal{H} \circ \mathcal{S}(T_1'')$ and $Q := \mathcal{H} \circ \mathcal{S}(T_2'')$, where points T_1'', T_2'' are as in GenericCodomainWith8Torsion, and theta coordinates of $I := (\alpha^{-1} : \beta^{-1} : \gamma^{-1} : \delta^{-1})$.

Output: A boolean `bool` indicating whether the 4-torsion points $\Phi(T_1''), \Phi(T_2'')$ are isotropic (i.e., $e_4(\Phi(T_1''), \Phi(T_2'')) = 1$).

$x_P, y_P, z_P, w_P \leftarrow P$
$x_Q, y_Q, z_Q, w_Q \leftarrow Q$
$\alpha^{-1}, \beta^{-1}, \gamma^{-1}, \delta^{-1} \leftarrow I$
$t_1 \leftarrow x_P \cdot \alpha^{-1}$
$t_2 \leftarrow y_P \cdot \beta^{-1}$
if $t_1 \neq t_2$ **then**
  return `false`
$t_1 \leftarrow z_P \cdot \gamma^{-1}$
$t_2 \leftarrow t_P \cdot \delta^{-1}$
if $t_1 \neq t_2$ **then**
  return `false`
$t_1 \leftarrow x_Q \cdot \alpha^{-1}$
$t_2 \leftarrow z_Q \cdot \gamma^{-1}$
if $t_1 \neq t_2$ **then**
  return `false`
$t_1 \leftarrow y_Q \cdot \beta^{-1}$
$t_2 \leftarrow t_Q \cdot \delta^{-1}$
if $t_1 \neq t_2$ **then**
  return `false`
return `true` // Total cost: at most $8\mathbf{M}$

(see Algorithm 8.40) method. We also use another faster method GluingEvalSpecial (see Algorithm 8.41) used when our point P is of the form $P = (P_1, 0)$ or $(0, P_2)$.

GluingEval takes as input the point $P \in E_1 \times E_2$, the 8-torsion point T_1'' lying above a generator of the kernel of Φ, together with the point $J = (x : x : y : y)$, the dual of the theta point $\Phi(T_1'')$, and the change-of-basis matrix that was computed during GluingCodomain, and returns the theta point $\Phi(P)$. It is detailed in Algorithm 8.40.

GluingEvalSpecial takes as input a point $P \in E_1 \times E_2$ of the form $(P_1, 0)$ or $(0, P_2)$ together with the point $I = (\alpha^{-1} : \beta^{-1} : \gamma^{-1} : 0)$, the "inverse" of the dual isogeneous theta null point over A and returns the theta point $\Phi(P)$. It uses the fact that, thanks to Algorithm 8.37, $W_{\varphi(P)} = 0$. It is detailed in Algorithm 8.41.

8.5.7. Splitting change of coordinates

We now look at how to compute a splitting $(2,2)$-isogeny $\Phi : A \to E_1 \times E_2$, where E_1, E_2 are elliptic curves and A is a PPAS. Such an isogeny is computed in two steps: (1) computing a 'generic' $(2,2)$-isogeny using GenericCodomainWith8Torsion, GenericCodomainWith4Torsion, or GenericCodomain to obtain an isogeny $\Phi : A \to B$ where $B \cong E_1 \times E_2$; (2) computing the isomorphism whose action on 0_B gives back the product theta structure $E_1 \times E_2$. We give an algorithm, called SplittingIsomorphism, for the latter step below in Algorithm 8.43. It calls a sub-algorithm GetIndexSplitting to determine the index (i, j) such that $U_{i,j}(0) = 0$, where $U_{i,j}$ are *level 4* theta coordinates. To compute the value of $U_{i,j}$, we require a function χ.[1] For our purposes, it suffices to know the output of χ on indices $(i, j) \in \{0, 1, 2, 3\}^2$:

$$\chi(i,j) := \begin{cases} 1, & \text{if } (i,j) \in \{(0,0),(0,1),(0,2),(0,3),(1,0),(1,2),(2,0),(2,1),(3,0),(3,3)\} \\ -1, & \text{if } (i,j) \in \{(1,1),(1,3),(2,2),(2,3),(3,1),(3,2)\} \end{cases}.$$

For more details on the theory and correctness of the splitting algorithm, we refer to [DMPR24]. In the case of Algorithm 8.48, thanks to the extra torsion points above our kernel, the theta structure on the codomain is set up so that we are always in the case of $(i, j) = (00, 11)$ of Algorithm 8.43.

Conversely to gluing, after computing a splitting $(2,2)$-isogeny $\Phi : A \to B$ such that $B \cong E_1 \times E_2$, we want to recover the Montgomery coefficients of E_1, E_2 from the theta null point 0_B of B. We depict how this is done in

[1] We use the notation χ as it is a *character*.

Algorithm 8.33 GenericCodomainWith4Torsion$(T_1', 0_A)$

Input: Theta coordinates of T_1' of order 4 such that $[2]T_1' \in \ker(\Phi)$ and the theta null point $0_A := (a : b : c : d)$ of A.

Output: Dual isogeneous theta null point $(\alpha : \beta : \gamma : \delta)$, the inverse of the dual isogeneous theta null point $(\alpha^{-1} : \beta^{-1} : \gamma^{-1} : \delta^{-1})$, and the theta null point 0_B on B. // Case $\alpha \cdot \beta \cdot \gamma \cdot \delta \neq 0$.

$(x\alpha\beta, \_, x\gamma\delta, \_) \leftarrow \mathcal{H} \circ \mathcal{S}(x_{T_1'}, y_{T_1'}, z_{T_1'}, w_{T_1'})$
$(\alpha^2, \beta^2, \gamma^2, \delta^2) \leftarrow \mathcal{H} \circ \mathcal{S}(a, b, c, d)$
$\alpha\beta \leftarrow$ SquareRoot$(\alpha^2 \cdot \beta^2)$
$\alpha\gamma \leftarrow$ SquareRoot$(\alpha^2 \cdot \gamma^2)$
$\beta \leftarrow \alpha\beta \cdot \alpha\gamma$
$\delta^{-1} \leftarrow \beta \cdot x\gamma\delta$
$\beta \leftarrow \beta \cdot x\alpha\beta$
$\delta \leftarrow x\gamma\delta \cdot \alpha\beta \cdot \alpha^2$
$\alpha \leftarrow x\alpha\beta \cdot \alpha^2$
$\gamma \leftarrow \alpha \cdot \gamma^2$
$\alpha \leftarrow \alpha \cdot \alpha\gamma$
$\alpha^{-1} \leftarrow x\alpha\beta \cdot \delta^2$
$\gamma^{-1} \leftarrow \alpha^{-1} \cdot \beta^2$
$\alpha^{-1} \leftarrow \alpha^{-1} \cdot \gamma^2$
$\beta^{-1} \leftarrow \alpha^{-1} \cdot \alpha\beta$
$\alpha^{-1} \leftarrow \alpha^{-1} \cdot \beta^2$
$\gamma^{-1} \leftarrow \gamma^{-1} \cdot \alpha\gamma$
$\delta^{-1} \leftarrow \delta^{-1} \cdot \beta^2$
$(a_2, b_2, c_2, d_2) \leftarrow \mathcal{H}(\alpha, \beta, \gamma, \delta)$
$0_B \leftarrow (a_2, b_2, c_2, d_2)$
return $(\alpha, \beta, \gamma, \delta), (\alpha^{-1}, \beta^{-1}, \gamma^{-1}, \delta^{-1}), 0_B$ // Total cost: $8\mathbf{S} + 17\mathbf{M} + 24\mathbf{a} + 2\mathbf{Sqrt}$

Algorithm 8.34 GenericCodomain(0_A)

Input: The theta constants $0_A := (a : b : c : d)$ of A.

Output: Dual isogeneous theta null point $(\alpha : \beta : \gamma : \delta)$, the inverse of the dual isogeneous theta null point $(\alpha^{-1} : \beta^{-1} : \gamma^{-1} : \delta^{-1})$, and the theta null point 0_B on B. // Case $\alpha \cdot \beta \cdot \gamma \cdot \delta \neq 0$.

$(\alpha^2, \beta^2, \gamma^2, \delta^2) \leftarrow \mathcal{H} \circ \mathcal{S}(a, b, c, d)$
$\alpha \leftarrow \alpha^2$
$\beta \leftarrow \alpha^2 \cdot \beta^2$
$\gamma \leftarrow \alpha^2 \cdot \gamma^2$
$\delta \leftarrow \alpha^2 \cdot \delta^2$
$\beta \leftarrow$ SquareRoot(β)
$\gamma \leftarrow$ SquareRoot(γ)
$\delta \leftarrow$ SquareRoot(δ)
$\alpha^{-1} \leftarrow \gamma^2 \cdot \delta^2$
$\beta^{-1} \leftarrow \alpha^{-1} \cdot \beta$
$\alpha^{-1} \leftarrow \alpha^{-1} \cdot \beta^2$
$\gamma^{-1} \leftarrow \delta^2 \cdot \beta^2 \cdot \gamma$
$\delta^{-1} \leftarrow \gamma^2 \cdot \beta^2 \cdot \delta$
$(a_2, b_2, c_2, d_2) \leftarrow \mathcal{H}(\alpha, \beta, \gamma, \delta)$
$0_B \leftarrow (a_2, b_2, c_2, d_2)$
return $(\alpha, \beta, \gamma, \delta), (\alpha^{-1}, \beta^{-1}, \gamma^{-1}, \delta^{-1}), 0_B$ // Total cost: $4\mathbf{S} + 10\mathbf{M} + 24\mathbf{a} + 3\mathbf{Sqrt}$

ThetaToProduct, which follows the discussion in Section 8.5.1.3. We also need to convert our points from theta product coordinates to Montgomery coordinates, which is done in Algorithm 8.46.

The Montgomery coordinates obtained after the computation of the entire $(2, 2)$-isogeny chain may still reveal some information about the one-dimensional isogenies underlying the isogeny matrix. To avoid any potential leakage, during signing, the matrix output of SplittingIsomorphism is randomized using the following strategy.

Algorithm 8.35 GenericEval(P, I)

Input: Theta coordinates of P and the inverse of the dual isogeneous theta null point $I := (\alpha^{-1} : \beta^{-1} : \gamma^{-1} : \delta^{-1})$ on B.
Output: Theta coordinates of $P' := \Phi(P)$. // Case $\beta \cdot \gamma \cdot \delta \neq 0$

$x_P, y_P, z_P, w_P \leftarrow P$
$\alpha^{-1}, \beta^{-1}, \gamma^{-1}, \delta^{-1} \leftarrow I$
$(X_P, Y_P, Z_P, W_P) \leftarrow \mathcal{H} \circ \mathcal{S}(x_P, y_P, z_P, w_P)$
$X' \leftarrow \alpha^{-1} X_P$
$Y' \leftarrow \beta^{-1} \cdot Y_P$
$Z' \leftarrow \gamma^{-1} \cdot Z_P$
$W' \leftarrow \delta^{-1} \cdot W_P$
$x_{P'}, y_{P'}, z_{P'}, w_{P'} \leftarrow \mathcal{H}(X', Y', Z', W')$
$P' \leftarrow (x_{P'} : y_{P'} : z_{P'} : w_{P'})$
return P' // Total cost: $4\mathbf{S} + 4\mathbf{M} + 16\mathbf{a}$

Algorithm 8.36 ActionByTranslation(P, Q)

Input: Four-torsion points $P = (P_1, P_2)$ and $Q = (Q_1, Q_2)$ on $E_1 \times E_2$.
Output: Array `mats` containing 2×2 matrices giving the action-of-translation by $P_1, Q_1 \in E_1$ and $P_2, Q_2 \in E_2$.

$P' := (P_1', P_2') \leftarrow [2]P$ // Cost: $4\mathbf{S} + 8\mathbf{M} + 8\mathbf{a}$ using xDBL
$Q' := (Q_1', Q_2') \leftarrow [2]Q$ // Cost: $4\mathbf{S} + 8\mathbf{M} + 8\mathbf{a}$ using xDBL
Write $P_i' = (X_i^{(P)} : Z_i^{(P)})$ and $Q_i' = (X_i^{(Q)} : Z_i^{(Q)})$ for $i = 1, 2$
Write $P_i = (U_i^{(P)} : W_i^{(P)})$ and $Q_i = (U_i^{(Q)} : W_i^{(Q)})$ for $i = 1, 2$
for i **from** 1 **up to** 2 **do**
  $WX_i^{(P)}, WZ_i^{(P)}, UX_i^{(P)}, UZ_i^{(P)} \leftarrow W_i^{(P)} \cdot Z_i^{(P)}, W_i^{(P)} \cdot Z_i^{(P)}, U_i^{(P)} \cdot X_i^{(P)}, U_i^{(P)} \cdot Z_i^{(P)}$
  $\delta_i^{(Q)} \leftarrow WX_i^{(Q)} - UZ_i^{(Q)}$
  $WX_i^{(Q)}, WZ_i^{(Q)}, UX_i^{(Q)}, UZ_i^{(Q)} \leftarrow W_i^{(Q)} \cdot Z_i^{(Q)}, W_i^{(Q)} \cdot Z_i^{(Q)}, U_i^{(Q)} \cdot X_i^{(Q)}, U_i^{(Q)} \cdot Z_i^{(Q)}$
  $\delta_i^{(Q)} \leftarrow WX_i^{(Q)} - UZ_i^{(Q)}$
Compute the inverses of $\delta_1^{(P)}, \delta_2^{(P)}, \delta_1^{(Q)}, \delta_2^{(Q)}, Z_1^{(P)}, Z_2^{(P)}, Z_1^{(Q)}, Z_2^{(Q)}$ using batched inversions // Cost: $21\mathbf{M} + 1\mathbf{I}$
Initialize array `mats` to store matrices
`pts` $\leftarrow [P, Q]$
for i **from** 1 **up to** 2 **do**
  for j **from** 1 **up to** 2 **do**
    $R \leftarrow$ `pts`$[j]$
    $M_{0,0} \leftarrow -U_i^{(R)} Z_i^{(R)} \cdot (\delta_i^{(R)})^{-1}$
    $M_{0,1} \leftarrow -W_i^{(R)} Z_i^{(R)} \cdot (\delta_i^{(R)})^{-1}$
    $M_{1,0} \leftarrow U_i^{(R)} X_i^{(R)} \cdot (\delta_i^{(R)})^{-1} - X_i^{(R)} \cdot (Z_i^{(R)})^{-1}$
    $M_{1,1} \leftarrow -M_{0,0}$
    $\mathbf{M} \leftarrow (M_{i,j})_{0 \leq i,j \leq 1}$
    Append $\mathbf{M}$ to `mats`
return `mats` // Total cost: $8\mathbf{S} + 69\mathbf{M} + 1\mathbf{I} + 24\mathbf{a}$

For each Montgomery curve E, there exist six isomorphic Montgomery curves. Given a theta-null point for E, we can obtain any of the other theta-null points corresponding to the other five isomorphic curves by applying a change of variables. For efficiency, we perform this change of variables to both the codomain curves obtained after applying SplittingIsomorphism. Concretely, this boils down to multiplying the matrix output by SplittingIsomorphism by one of the six precomputed change-of-variables matrices. This procedure is summarized in RandomizedProduct.

Algorithm 8.37 ThetaChangeOfBasis(P, Q)

Input: The points $P = (P_1, P_2)$ and $Q = (Q_1, Q_2)$ in the four torsion of $E_1 \times E_2$ such that the kernel for the gluing isogeny is $\langle [2]P, [2]Q \rangle$.
Output: The 4×4 change of basis matrix $\mathbf{N}$.

if P_1, P_2, Q_1, Q_2 do not have order 4 or $[2]P_1 = [2]P_2$ or $2]Q_1 = [2]Q_2$ **then**
  raise ("ThetaChangeOfBasis failed: gluing kernel is diagonal or not isotropic.")
$\mathbf{G}, \mathbf{G}', \mathbf{H}, \mathbf{H}' \leftarrow$ ActionByTranslation(P, Q) // Cost: $4\mathbf{S} + 61\mathbf{M} + 1\mathbf{I} + 16\mathbf{a}$
$t_1 \leftarrow \mathbf{G}_{0,0} \cdot \mathbf{H}_{0,0} + \mathbf{G}_{0,1} \cdot \mathbf{H}_{1,0}$
$t_2 \leftarrow \mathbf{G}_{1,0} \cdot \mathbf{H}_{0,0} + \mathbf{G}_{1,1} \cdot \mathbf{H}_{1,0}$
$t_3 \leftarrow \mathbf{G}'_{0,0} \cdot \mathbf{H}'_{0,0} + \mathbf{G}'_{0,1} \cdot \mathbf{H}'_{1,0}$
$t_4 \leftarrow \mathbf{G}'_{1,0} \cdot \mathbf{H}'_{0,0} + \mathbf{G}'_{1,1} \cdot \mathbf{H}'_{1,0}$
$N_{0,0} \leftarrow \mathbf{G}_{0,0} \cdot \mathbf{G}'_{0,0} + \mathbf{H}_{0,0} \cdot \mathbf{H}'_{0,0} + t_1 \cdot t_3 + 1$
$N_{0,1} \leftarrow \mathbf{G}_{0,0} \cdot \mathbf{G}'_{1,0} + \mathbf{H}_{0,0} \cdot \mathbf{H}'_{1,0} + t_1 \cdot t_4$
$N_{0,2} \leftarrow \mathbf{G}_{1,0} \cdot \mathbf{G}'_{0,0} + \mathbf{H}_{1,0} \cdot \mathbf{H}'_{0,0} + t_2 \cdot t_3$
$N_{0,3} \leftarrow \mathbf{G}_{1,0} \cdot \mathbf{G}'_{1,0} + \mathbf{H}_{1,0} \cdot \mathbf{H}'_{1,0} + t_2 \cdot t_4$
$N_{1,0} \leftarrow \mathbf{H}'_{0,0} \cdot N_{0,0} + \mathbf{H}'_{0,1} \cdot N_{0,1}$
$N_{1,1} \leftarrow \mathbf{H}'_{1,0} \cdot N_{0,0} + \mathbf{H}'_{1,1} \cdot N_{0,1}$
$N_{1,2} \leftarrow \mathbf{H}'_{0,0} \cdot N_{0,2} + \mathbf{H}'_{0,1} \cdot N_{0,3}$
$N_{1,3} \leftarrow \mathbf{H}'_{1,0} \cdot N_{0,2} + \mathbf{H}'_{1,1} \cdot N_{0,3}$
$N_{2,0} \leftarrow \mathbf{G}_{0,0} \cdot N_{0,0} + \mathbf{G}_{0,1} \cdot N_{0,2}$
$N_{2,1} \leftarrow \mathbf{G}_{0,0} \cdot N_{0,1} + \mathbf{G}_{0,1} \cdot N_{0,3}$
$N_{2,2} \leftarrow \mathbf{G}_{1,0} \cdot N_{0,0} + \mathbf{G}_{1,1} \cdot N_{0,2}$
$N_{2,3} \leftarrow \mathbf{G}_{1,0} \cdot N_{0,1} + \mathbf{G}_{1,1} \cdot N_{0,3}$
$N_{3,0} \leftarrow \mathbf{G}_{0,0} \cdot N_{1,0} + \mathbf{G}_{0,1} \cdot N_{1,2}$
$N_{3,1} \leftarrow \mathbf{G}_{0,0} \cdot N_{1,1} + \mathbf{G}_{0,1} \cdot N_{1,3}$
$N_{3,2} \leftarrow \mathbf{G}_{1,0} \cdot N_{1,0} + \mathbf{G}_{1,1} \cdot N_{1,2}$
$N_{3,3} \leftarrow \mathbf{G}_{1,0} \cdot N_{1,1} + \mathbf{G}_{1,1} \cdot N_{1,3}$
$\mathbf{N} \leftarrow (N_{i,j})_{0 \le i,j \le 3}$
return $\mathbf{N}$ // Total cost: $8\mathbf{S} + 113\mathbf{M} + 1\mathbf{I} + 49\mathbf{a}$

Algorithm 8.38 ProductToTheta$(\texttt{pts}, \mathbf{N})$

Input: List of points `pts`, where for $P \in$ `pts` we have $P := (P_1, P_2) \in E_1 \times E_2$, and $\mathbf{N}$ the change-of-basis matrix computed using ThetaChangeOfBasis.
Output: The corresponding point P' in theta coordinates on $A \cong E_1 \times E_2$.

`eval_pts` $\leftarrow [\,]$
$L \leftarrow \#$`pts`
for j **from** 1 **up to** L **do**
  $P = (P_1, P_2) \leftarrow$ `pts`$[j]$
  Write $P_i = (\theta_{0,i} : \theta_{1,i})$ for $i = 1, 2$.
  $x \leftarrow \theta_{0,1} \cdot \theta_{0,2}$
  $y \leftarrow \theta_{0,1} \cdot \theta_{1,2}$
  $z \leftarrow \theta_{1,1} \cdot \theta_{0,2}$
  $w \leftarrow \theta_{1,1} \cdot \theta_{1,2}$
  $P' \leftarrow \mathbf{N} \cdot (x : y : z : w)$ // Cost: at most $20\mathbf{M} + 12\mathbf{a}$
  Append P' to `eval_pts`
return `eval_pts` // Total cost: $+ 20k\mathbf{M} + 12k\mathbf{a}$ with $k = \#$`pts`

8.5.8. Computing a $(2, 2)$-isogeny chain between products of elliptic curves

We are now ready to describe the computation of a chain of $(2, 2)$-isogenies. In Isogeny22Chain (see Algorithm 8.47), we present the algorithm to compute a $(2, 2)$-isogeny chain between products of elliptic curves $\Phi : E_1 \times E_2 \to E_3 \times E_4$ using a balanced strategy (adjusted to account for the expensive gluing step at the start of the chain), on input points $P, Q \in E_1 \times E_2$ of order 2^e.

Algorithm 8.39 GluingCodomain(T_1'', T_2'')

Input: 8-torsion points $T_1'', T_2'' \in E_1 \times E_2$, and $\ker(\Phi) = \langle [4]T_1'' \rangle \oplus \langle [4]T_2'' \rangle$.

Output: Dual isogeneous theta null point $(\alpha : \beta : \gamma : 0)$, the "inverse" of the dual isogeneous theta null point $(\alpha^{-1} : \beta^{-1} : \gamma^{-1} : 0)$, the theta null point 0_A on A, the dual isogeneous theta point $\Phi(T_1'')$ on A, and the change-of-basis matrix $\mathbf{N}$ (re-used for point evaluation). // Case $\delta = 0$

$T_1' \leftarrow [2](T_1'')$ // Cost: $2\mathbf{S} + 4\mathbf{M} + 4\mathbf{a}$ using xDBL
$T_2' \leftarrow [2](T_2'')$ // Cost: $2\mathbf{S} + 4\mathbf{M} + 4\mathbf{a}$ using xDBL
$\mathbf{N} \leftarrow$ ThetaChangeOfBasis(T_1', T_2') // Total cost: $8\mathbf{S} + 113\mathbf{M} + 1\mathbf{I} + 49\mathbf{a}$
$[P_1, P_2] \leftarrow$ ProductToTheta$([T_1'', T_2''], \mathbf{N})$ // Total cost: $40\mathbf{M} + 24\mathbf{a}$
$x_1, y_1, z_1, t_1 \leftarrow P_1$
$x_2, y_2, z_2, t_2 \leftarrow P_2$
$(X_1, Y_1, Z_1, T_1) \leftarrow \mathcal{H} \circ \mathcal{S}(x_1, y_1, z_1, t_1)$
$(X_2, Y_2, Z_2, T_2) \leftarrow \mathcal{H} \circ \mathcal{S}(x_2, y_2, z_2, t_2)$
if $T_1 \neq 0$ or $T_2 \neq 0$ **then**
  raise ("GluingCodomain failed: last coordinate is non-zero")
if $X_1, X_2, Y_1, Z_2 = 0$ **then**
  raise ("GluingCodomain failed: projective factors are zero")
$\alpha \leftarrow X_1 \cdot X_2$
$\beta \leftarrow Y_1 \cdot X_2$
$\gamma \leftarrow X_1 \cdot Z_2$
$\alpha^{-1} \leftarrow Y_1 \cdot Z_2$
$\beta^{-1} \leftarrow \gamma$
$\gamma^{-1} \leftarrow \beta$
$x \leftarrow X_1 \cdot \alpha^{-1}$
$y \leftarrow Z_1 \cdot \gamma^{-1}$
$J \leftarrow (x : x : y : y)$
if $(Y_1 \cdot \beta^{-1} \neq x)$ **or** $(X_2 \cdot \alpha^{-1} \neq Y_2 \cdot \beta^{-1})$ **then**
  raise Exception: ("GluingCodomain failed: $[2]T_1''$, $[2]T_2''$ not isotropic") // Check only done in verification.
$(a_2, b_2, c_2, d_2) \leftarrow \mathcal{H}(\alpha, \beta, \gamma, 0)$
$0_A \leftarrow (a_2 : b_2 : c_2 : d_2)$
return $(\alpha : \beta : \gamma : 0), (\alpha^{-1} : \beta^{-1}, \gamma^{-1} : 0), 0_A, J, \mathbf{N}$ // Total cost (without checks): $12\mathbf{S} + 159\mathbf{M} + 1\mathbf{I} + 105\mathbf{a}$

Algorithm 8.40 GluingEval$(P, T_1'', J, \mathbf{N})$

Input: The point $P \in E_1 \times E_2$ and T_1'' a 8-torsion point such that $[4]T_1'' \in \ker(\Phi)$ both in Jacobian coordinates and the point $J = (x : x : y : y)$, the dual of the theta point $\Phi(T_1'')$ on A, and the change-of-basis matrix $\mathbf{N}$ computed during GluingCodomain.

Output: Theta coordinates of $\Phi(P)$.

$P_1, P_2 \leftarrow P$
$T_1, T_2 \leftarrow T_1''$
$x, y \leftarrow J$
$u_1, v_1, z_1 \leftarrow$ ADDComponents(P_1, T_1, E_1) // Total cost: $11\mathbf{M} + 5\mathbf{S} + 7\mathbf{a}$
$u_2, v_2, z_2 \leftarrow$ ADDComponents(P_2, T_2, E_2) // Total cost: $11\mathbf{M} + 5\mathbf{S} + 7\mathbf{a}$
$U \leftarrow (u_1 \cdot u_2 + v_1 \cdot v_2, u_1 \cdot w_1, w_1 \cdot u_2, w_1 \cdot w_2)$
$V \leftarrow (v_1 \cdot u_2 + u_1 \cdot v_2, v_1 \cdot w_1, w_1 \cdot v_2, 0)$
$U \leftarrow \mathbf{N} \cdot U$
$V \leftarrow \mathbf{N} \cdot V$
$U \leftarrow \mathcal{S}(U)$
$V \leftarrow \mathcal{S}(V)$
$X_\pm, Y_\pm, Z_\pm, W_\pm \leftarrow \mathcal{H}(U - V)$
$X_{\Phi(P)}, Y_{\Phi(P)}, Z_{\Phi(P)}, W_{\Phi(P)} \leftarrow X_\pm \cdot y, Y_\pm \cdot y, Z_\pm \cdot x, W_\pm \cdot x$
$(x_{\Phi(P)}, y_{\Phi(P)}, z_{\Phi(P)}, w_{\Phi(P)}) \leftarrow \mathcal{H}(X_{\Phi(P)}, Y_{\Phi(P)}, Z_{\Phi(P)}, W_{\Phi(P)})$
return $(x_{\Phi(P)} : y_{\Phi(P)} : z_{\Phi(P)} : w_{\Phi(P)})$ // Total cost: $62\mathbf{M} + 18\mathbf{S} + 50\mathbf{a}$

Algorithm 8.41 GluingEvalSpecial$(P, I, \mathbf{N})$

Input: The point $P \in E_1 \times E_2$ of the form $(P_1, 0)$ or $(0, P_2)$, the "inverse" of the dual isogeneous theta null point $I := (\alpha^{-1} : \beta^{-1} : \gamma^{-1} : 0)$ on A and the change-of-basis matrix $\mathbf{N}$ computed during GluingCodomain.
Output: Theta coordinates of $\Phi(P)$.

$P \leftarrow$ ProductToTheta$(P, \mathbf{N})$ // $20\mathbf{M} + 12\mathbf{a}$
$x_P, y_P, z_P, t_P \leftarrow P$
$\alpha^{-1}, \beta^{-1}, \gamma^{-1}, \_ \leftarrow I$
$(X_P, Y_P, Z_P, 0) \leftarrow \mathcal{H} \circ \mathcal{S}(x_P, y_P, z_P, w_P)$
$(X_{\Phi(P)}, Y_{\Phi(P)}, Z_{\Phi(P)}) \leftarrow X_P \cdot \alpha^{-1}, Y_P \cdot \beta^{-1}, Z_P \cdot \gamma^{-1}$
$(x_{\Phi(P)}, y_{\Phi(P)}, z_{\Phi(P)}, w_{\Phi(P)}) \leftarrow \mathcal{H}(X_{\Phi(P)}, Y_{\Phi(P)}, Z_{\Phi(P)}, 0)$
return $(x_{\Phi(P)} : y_{\Phi(P)} : z_{\Phi(P)} : w_{\Phi(P)})$ // Total cost: $4\mathbf{S} + 23\mathbf{M}$

Algorithm 8.42 GetIndexSplitting(0_A)

Input: Theta null point 0_A of theta structure $A \cong E_1 \times E_2$.
Output: Index (i, j) such that $U_{i,j}(0) = 0$.

$\texttt{inds} \leftarrow \{(0,0), (0,1), (0,2), (0,3), (1,0), (1,2), (2,0), (2,1), (3,0), (3,3)\}$
for k **from** 1 **up to** $\#\texttt{inds}$ **do**
  $(i, j) \leftarrow \texttt{inds}[k]$
  $U \leftarrow \sum_{k=0}^{3} \chi(i, k) \cdot 0_A[k] \cdot 0_A[j \oplus k]$ // Here, $\oplus$ denotes bitwise XOR.
  if $U = 0$ **then**
    return (i, j)

The algorithm Isogeny22ChainWithTorsion given by Algorithm 8.48, now takes input isotropic points $P, Q \in E_1 \times E_2$ of order 2^{e+2}. As such, the $(2, 2)$-isogeny chain can be computed without square roots.

REMARK 2. The first output of Isogeny22Chain or Isogeny22ChainWithTorsion, is the codomain product $E_3 \times E_4$. In the context of SQISIGN, E_3 can always be taken to be the commitment curve. This is due to our choice of theta structure, as well as the choice of splitting matrix output by SplittingIsomorphism. We remark that, for example, in Chapter 4 this was implicitly assumed in Line 29 of Algorithm 4.9.

Algorithm 8.43 SplittingIsomorphism(0_A)

Input: Theta null point 0_A of $A \cong E_1 \times E_2$.

Output: Isomorphism given by matrix $\mathbf{M}$, whose action on 0_A gives back the theta null point associated with the product theta structure.

$(i,j) \leftarrow$ GetIndexSplitting(0_A)
if $(i,j) = (00,00)$ **then**
  $\mathbf{M} \leftarrow \begin{pmatrix} 1 & \sqrt{-1} & 1 & \sqrt{-1} \\ 1 & -\sqrt{-1} & -1 & \sqrt{-1} \\ 1 & \sqrt{-1} & -1 & -\sqrt{-1} \\ -1 & \sqrt{-1} & -1 & \sqrt{-1} \end{pmatrix}$
else if $(i,j) = (01,00)$ **then**
  $\mathbf{M} \leftarrow \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & -1 & 1 \\ 1 & 1 & -1 & -1 \\ -1 & 1 & -1 & 1 \end{pmatrix}$
else if $(i,j) = (10,00)$ **then**
  $\mathbf{M} \leftarrow \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & -1 & -1 & 1 \\ -1 & -1 & 1 & 1 \end{pmatrix}$
else if $(i,j) = (11,00)$ **then**
  $\mathbf{M} \leftarrow \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ -1 & 1 & 1 & -1 \end{pmatrix}$
else if $(i,j) = (00,01)$ **then**
  $\mathbf{M} \leftarrow \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & -1 & 0 & 0 \end{pmatrix}$
else if $(i,j) = (10,01)$ **then**
  $\mathbf{M} \leftarrow \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & 1 & -1 & -1 \end{pmatrix}$
else if $(i,j) = (00,10)$ **then**
  $\mathbf{M} \leftarrow \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & -1 & 0 \end{pmatrix}$
else if $(i,j) = (01,10)$ **then**
  $\mathbf{M} \leftarrow \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$
else if $(i,j) = (00,11)$ **then**
  $\mathbf{M} \leftarrow \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix}$
else if $(i,j) = (11,11)$ **then**
  $\mathbf{M} \leftarrow \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$
$\mathbf{M} \leftarrow$ RandomizedProduct$(\mathbf{M})$ // Only done in signing.
return $\mathbf{M}$

Algorithm 8.44 RandomizedProduct($\mathbf{M}$)

Input: The splitting matrix $\mathbf{M}$
Output: The randomized matrix $\mathbf{N}$

1: $\mathcal{M} \leftarrow \left\{ \begin{pmatrix} 1&0&0&0\\0&1&0&0\\0&0&1&0\\0&0&0&1 \end{pmatrix}, \begin{pmatrix} 0&0&0&1\\0&0&1&0\\0&1&0&0\\1&0&0&0 \end{pmatrix}, \begin{pmatrix} 1&1&1&1\\1&-1&1&-1\\1&1&-1&-1\\1&-1&-1&1 \end{pmatrix}, \begin{pmatrix} 1&-1&-1&1\\-1&-1&1&1\\-1&1&-1&1\\1&1&1&1 \end{pmatrix}, \begin{pmatrix} -1&\sqrt{-1}&\sqrt{-1}&1\\\sqrt{-1}&-1&1&\sqrt{-1}\\\sqrt{-1}&1&-1&\sqrt{-1}\\1&\sqrt{-1}&\sqrt{-1}&-1 \end{pmatrix}, \begin{pmatrix} 1&\sqrt{-1}&\sqrt{-1}&-1\\\sqrt{-1}&1&-1&\sqrt{-1}\\\sqrt{-1}&-1&1&\sqrt{-1}\\-1&\sqrt{-1}&\sqrt{-1}&1 \end{pmatrix} \right\}$
2: Sample a random $\mathbf{N}$ in $\mathcal{M}$
3: $\mathbf{N} \leftarrow \mathbf{N} \cdot \mathbf{M}$
4: **return** $\mathbf{N}$

Algorithm 8.45 ThetaToProduct(0_A)

Input: The theta null point $0_A := (a : b : c : d)$ of PPAS A with theta product structure.
Output: The Montgomery coefficients of E_1, E_2, namely $(A_{24}^{(1)} : C_{24}^{(1)})$ and $(A_{24}^{(2)} : C_{24}^{(2)})$, such that $A \cong E_1 \times E_2$.

1: $(a, b, c, d) \leftarrow 0_A$
2: $t_1 \leftarrow ad$
3: $t_2 \leftarrow bc$
4: **if** $t_1 \neq t_2$ **then**
5: **raise** ("ThetaToProduct failed: 0_A does not come from a product theta structure")
6: $x \leftarrow a^4$
7: $y \leftarrow b^4$
8: $A_2 \leftarrow x + y$
9: $C_2 \leftarrow x - y$
10: $A_2 \leftarrow -2A_2$
11: $z \leftarrow c^4$
12: $A_1 \leftarrow x + z$
13: $C_1 \leftarrow x - z$
14: $A_1 \leftarrow -2A_1$
15: **if** $C_1 = 0$ **or** $C_2 = 0$ **then**
16: **raise** ("ThetaToProduct failed")
17: **return** $(A_1 : C_1), (A_2 : C_2)$

Algorithm 8.46 ThetaProductPointToMontgomery($P, 0_A$)

Input: A theta point $P := (x : y : z : t)$ and the theta null point $0_A := (a : b : c : d)$ of PPAS A with theta product structure.
Output: The Montgomery coordinates $(X(P_1) : Z(P_1))$ and $X(P_2) : Z(P_2)$ of $P = (P_1, P_2) \in E_1 \times E_2$.

1: $(a, b, c, d) \leftarrow 0_A$
2: $(x, y, z, t) \leftarrow P$
3: $X_1 \leftarrow a \cdot z + c \cdot x$
4: $Z_1 \leftarrow a \cdot z - c \cdot x$
5: $X_2 \leftarrow a \cdot y + b \cdot x$
6: $Z_2 \leftarrow a \cdot y - b \cdot x$
7: **return** $(X_1 : Z_1), (X_2 : Z_2)$

Algorithm 8.47 Isogeny22Chain$(P, Q, \texttt{pts})$

Input: Points $P, Q \in E_1 \times E_2$ of order 2^e, a positive integer e, and an array `pts` containing points on $E_1 \times E_2$ with at least one point zero.

Output: The codomain $E_3 \times E_4$ of the $(2,2)$-isogeny chain $\Phi = \Phi_e \circ \cdots \circ \Phi_1$ such that $\ker(\Phi) = \langle P, Q \rangle$ with $\deg(\Phi_k) = 2$ for all $k = 1, \ldots, e$, and the evaluated points $[\,\Phi(P) \mid P \in \texttt{pts}\,]$.

Initialize a list $\texttt{strat\_pts} \leftarrow [(P, Q)]$
Initialize a list $\texttt{orders} \leftarrow [e-2]$
$k \leftarrow 0$
while $\texttt{orders}[k] \neq 1$ **do**
  $k \leftarrow k + 1$
  if $\texttt{orders}[k-1] \geq 16$ **then**
    $n \leftarrow \lfloor \texttt{orders}[k-1]/2 \rfloor$
  else
    $n \leftarrow \texttt{orders}[k-1] - 1$
  $(R, S) \leftarrow \texttt{strat\_pts}[k-1]$
  for i **from** 0 **up to** $n-1$ **do**
    $R \leftarrow \mathsf{DBL}(R, A_1)$
    $S \leftarrow \mathsf{DBL}(S, A_2)$
  $\texttt{strat\_pts}[k] \leftarrow (R, S)$
  $\texttt{orders}[k] \leftarrow \texttt{orders}[k-1] - n$
$\_, I, 0_A, J, \mathbf{N} \leftarrow \mathsf{GluingCodomain}(\texttt{strat\_pts}[k])$ // Computing $\Phi_1 : E_1 \times E_2 \to A_1$
$\texttt{pts} \leftarrow [\mathsf{GluingEvalSpecial}(P, I, \mathbf{N}) \mid P \in \texttt{pts}]$
for i **from** 0 **up to** $k-1$ **do**
  $\texttt{strat\_pts}[i] \leftarrow \mathsf{GluingEval}(\texttt{strat\_pts}[i], I, J, \mathbf{N})$
  $\texttt{orders}[i] \leftarrow \texttt{orders}[i] - 1$
$k \leftarrow k - 1$
$\texttt{consts} \leftarrow \mathsf{ThetaPrecomp}(0_A)$
$j \leftarrow 0$
while $k \geq 0$ **and** $\texttt{orders}[k] \neq 0$ **do**
  $j \leftarrow j + 1$
  while $\texttt{orders}[k] \neq 1$ **do**
    $k \leftarrow k + 1$
    $n \leftarrow \lfloor \texttt{orders}[k-1]/2 \rfloor$
    for i **from** 0 **up to** $n-1$ **do**
      $\texttt{strat\_pts}[k] \leftarrow \mathsf{ThetaDBL}(\texttt{strat\_pts}[k-1], \texttt{consts})$
      $\texttt{orders}[k] \leftarrow \texttt{orders}[k-1] - n$
  $0'_A, I, 0_A \leftarrow \mathsf{GenericCodomainWith8Torsion}(\texttt{strat\_pts}[k])$ // Computing $\Phi_k : A_{k-1} \to A_k$
  $\texttt{pts} \leftarrow [\mathsf{GenericEval}(P, I) \mid P \in \texttt{pts}]$
  $\texttt{consts} \leftarrow \mathsf{ThetaPrecomp}(0_A, 0'_A)$
  for i **from** 0 **up to** $k-1$ **do**
    $\texttt{strat\_pts}[i] \leftarrow \mathsf{GenericEval}(\texttt{strat\_pts}[i], I)$
    $\texttt{orders}[j] \leftarrow \texttt{orders}[j] - 1$
  $k \leftarrow k - 1$
if $e \geq 3$ **then**
  $\texttt{strat\_pts}[0] \leftarrow \mathsf{GenericEval}(\texttt{strat\_pts}[0], I)$
$\_, I, 0_A \leftarrow \mathsf{GenericCodomainWith4Torsion}(\texttt{strat\_pts}[0], 0_A)$ // Computing $\Phi_{e-1} : A_{e-2} \to A_{e-1}$
$\texttt{pts} \leftarrow [\mathsf{GenericEval}(P, I) \mid P \in \texttt{pts}]$
$\_, I, 0_A \leftarrow \mathsf{GenericCodomain}(0_A)$ // Computing $\Phi_e : A_{e-1} \to A_e$
$\texttt{pts} \leftarrow [\mathsf{GenericEval}(P, I) \mid P \in \texttt{pts}]$
$\mathbf{M} \leftarrow \mathsf{SplittingIsomorphism}(0_A)$
$0_A \leftarrow \mathbf{M} \cdot 0_A$
$\texttt{pts} \leftarrow [\mathbf{M} \cdot P \mid P \in \texttt{eval\_pts}]$
$(A_3 : C_3), (A_4 : C_4) \leftarrow \mathsf{ThetaToProduct}(0_A)$
$\texttt{pts} \leftarrow [\mathsf{ThetaProductPointToMontgomery}(P, 0_A) \mid P \in \texttt{pts}]$
Let E_3, E_4 be the elliptic curves defined by Montgomery coefficients $(A_3 : C_3), (A_4 : C_4)$
return $E_3 \times E_4, \texttt{pts}$

Algorithm 8.48 Isogeny22ChainWithTorsion$(P, Q, \texttt{pts})$

Input: Jacobian points $P, Q \in E_1 \times E_2$ of order 2^{e+2}, Montgomery coefficients A_1, A_2 of E_1, E_2 respectively, a positive even integer e, and an array `pts` containing points on $E_1 \times E_2$ with at least one point zero.

Output: The codomain $E_3 \times E_4$ of the $(2,2)$-isogeny chain $\Phi = \Phi_e \circ \cdots \circ \Phi_1$ such that $\ker(\Phi) = \langle P, Q \rangle$ with $\deg(\Phi_k) = 2$ for all $k = 1, \ldots, e$, and the evaluated points $[\,\Phi(P) \mid P \in \texttt{pts}\,]$.

Initialize a list $\texttt{strat\_pts} \leftarrow [(P, Q)]$
Initialize a list $\texttt{orders} \leftarrow [e]$
$k \leftarrow 0$
while $\texttt{orders}[k] \neq 1$ **do**
  $k \leftarrow k + 1$
  if $\texttt{orders}[k-1] \geq 16$ **then**
    $n \leftarrow \lfloor \texttt{orders}[k-1]/2 \rfloor$
  else
    $n \leftarrow \texttt{orders}[k-1] - 1$
  $(R, S) \leftarrow \texttt{strat\_pts}[k-1]$
  for i **from** 0 **up to** $n-1$ **do**
    $R \leftarrow \mathsf{DBL}(R, A_1)$
    $S \leftarrow \mathsf{DBL}(S, A_2)$
  $\texttt{strat\_pts}[k] \leftarrow (R, S)$
  $\texttt{orders}[k] \leftarrow \texttt{orders}[k-1] - n$
$\_, I, 0_A, J, \mathbf{N} \leftarrow \mathsf{GluingCodomain}(\texttt{strat\_pts}[k])$ // Computing $\Phi_1 : E_1 \times E_2 \to A_1$
$\texttt{pts} \leftarrow [\mathsf{GluingEvalSpecial}(P, I, \mathbf{N}) \mid P \in \texttt{pts}]$
for i **from** 0 **up to** $k-1$ **do**
  $\texttt{strat\_pts}[i] \leftarrow \mathsf{GluingEval}(\texttt{strat\_pts}[i], I, J, \mathbf{N})$
  $\texttt{orders}[i] \leftarrow \texttt{orders}[i] - 1$
$k \leftarrow k - 1$
$\texttt{consts} \leftarrow \mathsf{ThetaPrecomp}(0_A)$
$j \leftarrow 0$
while $k \geq 0$ **and** $\texttt{orders}[k] \neq 0$ **do**
  $j \leftarrow j + 1$
  while $\texttt{orders}[k] \neq 1$ **do**
    $k \leftarrow k + 1$
    $n \leftarrow \lfloor \texttt{orders}[k-1]/2 \rfloor$
    for i **from** 0 **up to** $n-1$ **do**
      $\texttt{strat\_pts}[k] \leftarrow \mathsf{ThetaDBL}(\texttt{strat\_pts}[k-1], \texttt{consts})$
      $\texttt{orders}[k] \leftarrow \texttt{orders}[k-1] - n$
  $0'_A, I, 0_A \leftarrow \mathsf{GenericCodomainWith8Torsion}(\texttt{strat\_pts}[k])$ // Computing $\Phi_k : A_{k-1} \to A_k$
  $\texttt{pts} \leftarrow [\mathsf{GenericEval}(P, I) \mid P \in \texttt{pts}]$
  $\texttt{consts} \leftarrow \mathsf{ThetaPrecomp}(0_A, 0'_A)$
  for i **from** 0 **up to** $k-1$ **do**
    $\texttt{strat\_pts}[i] \leftarrow \mathsf{GenericEval}(\texttt{strat\_pts}[i], I)$
    $\texttt{orders}[j] \leftarrow \texttt{orders}[j] - 1$
  $k \leftarrow k - 1$
$\mathbf{M} \leftarrow \mathsf{SplittingIsomorphism}(0_A)$
$0_A \leftarrow \mathbf{M} \cdot 0_A$
$\texttt{pts} \leftarrow [\mathbf{M} \cdot P \mid P \in \texttt{eval\_pts}]$
$(A_3 : C_3), (A_4 : C_4) \leftarrow \mathsf{ThetaToProduct}(0_A)$
$\texttt{pts} \leftarrow [\mathsf{ThetaProductPointToMontgomery}(P, 0_A) \mid P \in \texttt{pts}]$
Let E_3, E_4 be the elliptic curves defined by Montgomery coefficients $(A_3 : C_3), (A_4 : C_4)$
return $E_3 \times E_4, \texttt{pts}$

CHAPTER 9

Heuristics and failure cases

The role of this chapter is to list and analyze possible failure cases of various algorithms involved in the signature process of SQISIGN. In particular, the goal of this chapter is to explain the heuristics behind the probability estimates that are used to choose the parameters.

9.1. Heuristics on lattices and ideals

The termination of a few algorithms related to ideal lattices relies on mild heuristics. The first one guarantees Algorithm LatticeSampling terminates in a constant number of steps on average:

Heuristic 9.1.1. The number of lattice points in an ellipsoid S centered at the origin is well approximated by the number of lattice points in a parallelogram $B \supset S$ multiplied by $\frac{\mathrm{vol}(S)}{\mathrm{vol}(B)}$.

The next one guarantees Algorithm IdealGenerator always terminates quickly:

Heuristic 9.1.2. Enumerating integer linear combinations of a quaternion ideal basis by increasing ℓ_1 norm finds a generator of the ideal in a small number of steps.

9.2. Chains of $(2,2)$-isogenies

During key generation, signing, and verifying we compute several chains of isogenies of the form

$$E_1 \times E_2 \xrightarrow{\Phi_1} A_1 \xrightarrow{\Phi_2} A_2 \quad \cdots \quad A_{e-2} \xrightarrow{\Phi_{e-1}} A_{e-1} \xrightarrow{\Phi_e} E_3 \times E_4.$$

The algorithms we gave in Section 8.5 are not universal and may fail in two possible ways. We argue that these failures happen with negligible probability during an honest execution, and thus may be ignored during key generation / signing (we nevertheless catch them and restart the process). If they happen during verification, with overwhelming probability they indicate a malicious signature and thus lead to rejection.

The first failure case happens if we encounter a splitting before the final step of a chain, i.e., if A_k has product theta structure for $1 \leq k < e$. This failure can never happen during verification of an honest signature. To bound the failure probability in key generation / signing, we make the following assumption:

Heuristic 9.2.1. The surfaces A_k encountered along the chains of $(2,2)$-isogenies computed in SQIsign.KeyGen and SQIsign.Sign behave like uniformly random superspecial PPAS.

The number of products of supersingular elliptic curves $E_1 \times E_2$ is $O(p^2)$, and the number of superspecial PPAS is $O(p^3)$, thus the heuristic ensures the computation of a $(2,2)$-isogeny chain fails with probability $\tilde{O}(p^{-1})$.

The second failure case happens, during the first gluing isogeny $E_1 \times E_2 \to A$, when the base change matrix we compute in Algorithm 8.37 is 0. This happens only when the trace of the coordinate $X_1 \cdot X_2$ under our kernel is zero, where $(X_1 : Z_1)$ and $(X_2 : Z_2)$ are the Montgomery coordinates on E_1 and E_2 respectively.

Heuristic 9.2.2. Let $E_1 \times E_2$ be a product of elliptic curves to which we apply a 2-dimensional isogeny computations during an honest execution of SQIsign.KeyGen, SQIsign.Sign, and SQIsign.Verify. Then the trace of the product coordinate $X_1 \cdot X_2$ under the gluing kernel, behaves like an independent random element of a $\mathbb{F}(p^2)$ vector space of dimension one.

Clearly the chance of encountering a zero trace is $O(p^{-2})$, and the computation only involves $O(1)$ two-dimensional isogenies: thus, the total failure probability for the second type of failures is in $O(p^{-2})$.

We note that this second failure may also happen (with negligible probability) during verification of an honest signature, because the gluing is computed from the other side in the verification compared to the signature.

9.3. Ideal-to-isogeny translation

The IdealToIsogeny algorithm is the main algorithmic building block behind SQIsign key generation and signature. It can fail in three places:

(1) If the call to GeneralizedRepresentInteger fails in one of the two executions of FixedDegreeIsogeny,
(2) If the call to SuitableIdeals fails,
(3) If any of the calls to Isogeny22Chain fail.

The latter was already discussed in Section 9.2 as it is not unique to the IdealToIsogeny algorithm.

There are also possible failures during the calls to RandomEquivalentPrimeIdeal in the Key Generation and commitment phases. As they are somewhat related to the possible failures of GeneralizedRepresentInteger, we treat both cases at once in Section 9.3.1. The failures of SuitableIdeals are treated in 9.3.2. A common heuristic used in all of them is the following:

Heuristic 9.3.1. Integers represented by a quadratic form behave like independent random integers of the same size with respect to factorization, coprimality and quadratic residuosity.

This type of assumption is very common in number theory, and after accounting for obstructions that may arise in specific cases (e.g., some congruence properties), tends to be satisfied in practice and can be easily confirmed experimentally.

Some of these assumptions could be removed assuming variants of the Generalized Riemann Hypothesis (GRH), following [Wes22], but the price to pay would be an increase of parameters and significant loss in efficiency. We do not pursue this direction in SQIsign.

9.3.1. GeneralizedRepresentInteger and RandomEquivalentPrimeIdeal

GeneralizedRepresentInteger is called in two places in the whole algorithm:

- In FixedDegreeIsogeny with input $M \approx p \cdot 2^{\texttt{QUAT\_repres\_bound\_input}}$, any of the orders $\mathcal{O}_t$, and `isogenyCond` set;
- In RandomIdealGivenNorm with input $M \approx p^{3/2}$, order $\mathcal{O}_0$ and `isogenyCond` unset.

We analyze the former as it is the most constrained call; the same arguments apply to the latter.

GeneralizedRepresentInteger calls Cornacchia to solve an equation of the form $x^2 + qy^2 = M'$ for $M' = 4M - p(z^2 + qw^2)$, where z, w are random in some intervals. This call succeeds if and only if:

- $-q$ is a square modulo M' and
- M' is a prime represented by $x^2 + qy^2$;

because `isogenyCond` is set, however, the solution may be discarded if it does not satisfy an additional constraint modulo 4.

Let $B = p \cdot 2^{\texttt{QUAT\_repres\_bound\_input}+2}$, assuming Heuristic 9.3.1 M' behaves like a random integer and thus we expect that it will be prime with probability $\approx 1/\ln(B)$ and that $-q$ will be a square with probability $1/2$. The additional modular constraint is satisfied with probability $1/16$.

Recalling the definition

$$\texttt{QUAT\_repres\_bound\_input} = \left\lceil \log_2(\log_{1-1/(64\log_2(p))}(2^{-64})) \right\rceil,$$

we see that $B < p^2$, and thus we can lower-bound the probability of success of one iteration of the main loop by

$$\frac{1}{64\ln(p)} > \frac{1}{64\log_2(p)}.$$

Now observe that the loop repeats up to $\lceil 4M/(p\sqrt{q}) \rceil \approx 2^{\texttt{QUAT\_repres\_bound\_input}}$ times, so the total failure probability is less than

$$\left(1 - \frac{1}{64\log_2(p)}\right)^{2^{\texttt{QUAT\_repres\_bound\_input}}} < 2^{-64}.$$

The analysis of RandomEquivalentPrimeIdeal is similar. There, we search for an equivalent ideal of prime norm to the ideal I in input. In practice, this means finding $\beta \in I$ such that $\mathrm{nrd}(\beta)/\mathrm{nrd}(I)$ is prime. Given a basis $\alpha_1, \alpha_2, \alpha_3, \alpha_4$ this is equivalent to finding a prime number represented by the quadratic form

$$q_I : (c_1, c_2, c_3, c_4) \mapsto \frac{\mathrm{nrd}(c_1\alpha_1 + c_2\alpha_2 + c_3\alpha_3 + c_4\alpha_4)}{\mathrm{nrd}(I)}.$$

We sample c_1, c_2, c_3, c_4 in a box of size $[-\texttt{QUAT\_equiv\_bound\_coeff}, \texttt{QUAT\_equiv\_bound\_coeff}]^4$ and so there are $(2\texttt{QUAT\_equiv\_bound\_coeff} + 1)^4$ different candidates.

Again, we use Heuristic 9.3.1 to argue that numbers represented by q_I behave as random integers of the same size. Since $\alpha_1, \alpha_2, \alpha_3, \alpha_4$ is L2-reduced we have

$$q_I(c_1, c_2, c_3, c_4) \approx \frac{8p}{\pi^2} \cdot \texttt{QUAT\_equiv\_bound\_coeff}^2$$

by [DLRW24, Lemma 48], so we can bound the probability that it is prime by $\frac{1}{2\log(p)}$. Then, proceeding like before we estimate the total failure probability to

$$\left(1 - \frac{1}{2\log(p)}\right)^{(2\texttt{QUAT\_equiv\_bound\_coeff}+1)^4} < 2^{-64}.$$

9.3.2. SuitableIdeals

The last heuristic algorithm in the ideal-to-isogeny translation is SuitableIdeals, and it is also the most difficult to analyze. We tackle it with a combination of heuristics, experimental data, and approximations. Its parameters are a list of $n_{\text{orders}} + 1$ special extremal orders $\mathcal{O}_i$ (see Section 3.1.7) and an integer `FINDUV_box_size` defining the search space. Their choice greatly affects the efficiency and the failure probability of the algorithm.

SuitableIdeals takes as input a left $\mathcal{O}_0$-ideal I_0 and starts by constructing the ideals $I_i = J_i I_0$ where J_i is a fixed ideal connecting O_i to O_0. Clearly the ideals I_i are correlated, but we make a first simplifying assumption:

Heuristic 9.3.2. Let I be an ideal connecting $\mathcal{O}_0$ to $\mathcal{O}$-ideal. Let $(\mathcal{O}_i)$ be a collection of special extremal orders and let J_i be ideals connecting $\mathcal{O}_i$ to $\mathcal{O}_0$. Then the ideal classes of $I_i = J_i I$ behave like independent random left ideal classes of $\mathcal{O}_i$.

This assumption is clearly false. For example, because all $\mathcal{O}_i$ are special extremal, the classes cannot all have representatives of too small norm (see [LB20]). However, in practice the behavior of the I_i's is quite close to that of random ideals, and, if anything, the correlation plays in favor of SuitableIdeals.

Then, SuitableIdeals defines quadratic forms $q_{I_i}(\alpha) = \operatorname{nrd}(\alpha)/\operatorname{nrd}(I_i)$ and enumerates elements from all I_i's until it finds a pair (β_s, β_t) such that:

(1) $q_{I_s}(\beta_s)$ and $q_{I_t}(\beta_t)$ are both odd and coprime;
(2) there exists a solution to the equation $uq_{I_s}(\beta_s) + vq_{I_t}(\beta_t) = 2^f$ with u and v positive.

Using Heuristic 9.3.1, we can treat $q_{I_s}(\beta_s)$ and $q_{I_t}(\beta_t)$ as random integers, then it is well known that the first condition is satisfied with probability

$$\frac{1}{3\zeta(2)} = \frac{2}{\pi^2}.$$

Granted the first condition, the second condition is much trickier: when $(q_{I_s}(\beta_s) - 1)(q_{I_t}(\beta_t) - 1) < 2^f + 1$ then we know u and v always exist. However when the product gets larger we may be out of luck. Here we introduce one more assumption:

Heuristic 9.3.3. Let $c > 0$ be fixed and let a, b be uniformly random positive coprime integers such that $ab > c$. The probability that there exist u, v positive such that $ua + vb = c$ is approximately c/ab.

Assuming this and 9.3.1, the probability that one pair (β_s, β_t) satisfies the constraints is

$$\frac{2}{\pi^2} \frac{2^f}{q_{I_s}(\beta_s) q_{I_t}(\beta_t)}. \tag{11}$$

We now analyze how large the denominator can be.

For each ideal I_i, SuitableIdeals first computes an L2-reduced basis $(\alpha_1, \alpha_2, \alpha_3, \alpha_4)$, then enumerates $\beta_i = c_1\alpha_1 + c_2\alpha_2 + c_3\alpha_3 + c_4\alpha_4$ in a parallelogram $|c_j| \leq \texttt{FINDUV\_box\_size}$. The size of β_i will depend on the norms of the α_j, which in turn depend on the minima of the lattice I_i. We make a simplifying assumption:

Heuristic 9.3.4. Let $(\alpha_1, \alpha_2, \alpha_3, \alpha_4)$ be an L2-reduced basis of a lattice of $B_{p,\infty}$ with quadratic form q, ordered by increasing norm. Then $q(\alpha_j)$ is close to the j-th minimum of the lattice.

Our experiments show that this assumption is verified in practice, and in fact that in a large majority of cases $q(\alpha_j)$ is the j-th minimum.

We need to understand the distribution of the minima. Let I be a left $\mathcal{O}$-ideal with Minkowski basis $(\alpha_1, \alpha_2, \alpha_3, \alpha_4)$. Write $\nu_i = q(\alpha_i)$. We have

$$\frac{p^2}{16} \leq \nu_1\nu_2\nu_3\nu_4 \leq \frac{p^2}{4},$$

where the lower bound is given by the volume of the lattice and the upper bound follows from [Cas78, Chapter 12]. As a consequence,

$$\nu_1 \leq \sqrt{p/2} \qquad \text{and} \qquad \nu_4 \geq \sqrt{p}/2.$$

Because we only consider special extremal orders $\mathcal{O}$ containing an element of norm q, it is always the case that $\nu_2 \leq \nu_1 q$. In fact, it is often the case that $\nu_4 \leq \nu_3 q$ too. To simplify, we shall ignore q and use in our estimates

$$\nu_1 \approx \nu_2 \qquad \text{and} \qquad \nu_3 \approx \nu_4.$$

Then

$$\frac{p}{4} \lessapprox \nu_1\nu_4 \lessapprox \frac{p}{2}.$$

Extensive experiments on $\mathcal{O}_0$-ideals show that $\nu_1\nu_4$ is closer to the lower bound with overwhelming probability, and especially so when ν_1 is far from $\sqrt{p/2}$, which is the case we are mostly interested in. Thus, we are going to make the approximation

$$\nu_4 \approx \frac{p}{4\nu_1}. \tag{12}$$

The distribution of ν_1 is relatively easy to understand. Say I is a random left $\mathcal{O}$-ideal, ν_1 is the norm of the smallest ideal equivalent to I. There are $\psi(n)$ left $\mathcal{O}$-ideals of norm n, where ψ is the Dedekind psi function

$$\psi(n) = n\prod_{p|n}(1 + 1/p),$$

thus there are

$$\sum_{x\leq n} \psi(x) \approx \frac{\zeta(2)}{2\zeta(4)}n^2 = \frac{15}{2\pi^2}n^2 \tag{13}$$

ideals of norm $\leq n$ (see [Apo76, Exercise 3.11]). The total number of ideal classes is $\approx p/12$, so if all ideals of norm $\leq n$ were in different classes we would expect

$$\Pr\{\nu_1 \leq n\} \approx \frac{90}{\pi^2}\left(\frac{n}{\sqrt{p}}\right)^2.$$

However, when $\mathcal{O}$ is special extremal it contains elements of small norm ω, thus if I has small norm so does $I\omega$, which is in the same class. Nevertheless, $\mathcal{O}$ contains *few* elements of small norm, thus we still expect $\Pr\{\nu_1 < n\}$ to grow with the square of n. Our experiments on $\mathcal{O}_0$ show an actual distribution of

$$\Pr\{\nu_1 < n\} \approx \frac{6.45652}{2}\left(\frac{n}{\sqrt{p}}\right)^2, \tag{14}$$

which is very accurate for n up to $\approx \sqrt{p}/2$. Combining Eq. (14) with Eq. (12) gives

$$\Pr\{\nu_4 > n\} \approx \frac{6.45652}{32}\left(\frac{\sqrt{p}}{n}\right)^2, \tag{15}$$

which, although less accurate, is still a pretty good approximation for the tail $n \gtrapprox \sqrt{p}$.

Finally, we come to the estimation of the failure rate. SuitableIdeals enumerates elements $\beta_i \in I_i$ in a box of sides $|c_j| \leq$ `FINDUV_box_size`, thus

$$q_{I_i}(\beta_i) \lessapprox \texttt{FINDUV\_box\_size}^2\nu_4^{(i)}.$$

Let us set a cutoff parameter $C > 0$ and overestimate the failure probability by analyzing a modified version of SuitableIdeals which discards any ideal I_i with $\nu_4 > C\sqrt{p}$ and ignores its β_i. This variant keeps $0 \leq s \leq 1+n_{\text{orders}}$ ideals with probability

$$F_1(C, s) = \binom{1 + n_{\text{orders}}}{s}\left(1 - \Pr\{\nu_4 > C\sqrt{p}\}\right)^s\left(\Pr\{\nu_4 > C\sqrt{p}\}\right)^{1+n_{\text{orders}}-s}. \tag{16}$$

Each of the kept ideals contributes

$$B = (2\,\texttt{FINDUV\_box\_size} + 1)^4 - 1 \tag{17}$$

TABLE 8. Estimated failure probabilities of SuitableIdeals for all security levels.

	FINDUV_box_size	$1 + n_{\text{orders}}$	$(p+1)/2^f$	$\operatorname{argmin} F(C)$	$\min F(C)$
NIST level I	2	7	5	18	2^{-65}
NIST level III	3	8	65	10	2^{-61}
NIST level V	3	7	27	13	2^{-60}

elements β_i to the search, for a total of $sB(sB-1)$ pairs to test. The probability that one of these pairs passes the test is given in Eq. (11), substituting $\texttt{FINDUV\_box\_size}^2 C\sqrt{p}$ for the norms we get a lower bound of

$$\frac{2 \cdot 2^f}{\texttt{FINDUV\_box\_size}^4 \pi^2 C^2 p}. \tag{18}$$

Putting everything together we obtain an estimate for the failure probability of

$$F(C) = \sum_{s=0}^{1+n_{\text{orders}}} F_1(C,s) \left(\frac{2 \cdot 2^f}{\texttt{FINDUV\_box\_size}^4 \pi^2 C^2 p} \right)^{sB(sB-1)}. \tag{19}$$

Finally, we look for the $C \in [1/2, \sqrt{p}]$ that minimizes $F(C)$. Substituting the constant defined in this document, we find values for FINDUV_box_size, n_{orders}, and C that achieve failure probability 2^{-60} or lower at all security levels, as reported in Table 8.

To gain more confidence in these estimates, we ran SuitableIdeals for values of $n_{\text{orders}} = 0, 1$, and we compared their failure rates with the predicted results. For level I parameters:

- For $n_{\text{orders}} = 0$ we observed 11 029 failures over 3 269 000 runs, i.e., a failure rate $2^{-8.2}$ against a predicted probability of 2^{-9};
- For $n_{\text{orders}} = 1$ we observed 6 failures over 20 553 000 runs, i.e., a failure rate $2^{-21.7}$ against a predicted probability of 2^{-18}.

For level III parameters:

- For $n_{\text{orders}} = 0$ we observed 15 022 failures over 1 162 000 runs, i.e., a failure rate $2^{-6.3}$ against a predicted probability of 2^{-7};
- For $n_{\text{orders}} = 1$ we observed 26 failures over 7 967 000 runs, i.e., a failure rate $2^{-18.2}$ against a predicted probability of 2^{-14}.

For level V parameters:

- For $n_{\text{orders}} = 0$ we observed 10 096 failures over 3 468 000 runs, i.e., a failure rate $2^{-8.4}$ against a predicted probability of 2^{-8};
- For $n_{\text{orders}} = 1$ we observed 4 failures over 4 429 000 runs, i.e., a failure rate 2^{-20} against a predicted probability of 2^{-16}.

These experiments give support the quality of our heuristic approximation and give us confidence in having reached the desired failure probability.

CHAPTER 10

Security analysis

This chapter discusses the security of SQIsign. In Section 10.1, we prove that SQIsign is existentially unforgeable against chosen message attacks (EUF-CMA) under two isogeny-based hardness assumptions. Then, in Section 10.2, we analyze the resistance of SQIsign to known attacks, which helps determine concrete parameters. Finally, in Section 10.3, we conclude with a discussion on various choices made for the parameters and how these choices have no known impact on security.

10.1. Security reductions

We analyze the security of SQIsign by relying on the *Fiat-Shamir with hints*[1] framework [ABDF+25]: we follow the usual Fiat-Shamir approach, where the soundness and zero-knowledge properties are proved separately. In this framework, however, the zero-knowledge simulator is given access to additional information, which we refer to as hints. The analysis in [ABDF+25] shows that such hint-assisted zero-knowledge property is sufficient to obtain EUF-CMA security for the resulting signature scheme, under a variant of the soundness assumption where the attacker has access to the hints. While in this section we give a high-level overview of the security proofs and its main results, we refer the reader to [ABDF+25] for a more thorough treatment. Note that we analyze a version of SQIsign where its subroutines do not fail: we address the impact of algorithm failures on the security of the protocol in Section 10.2.8.

Definition of EUF-CMA security. The EUF-CMA security of a signature scheme S is formulated as a game between an adversary $\mathcal{A}$ and a challenger.

Definition 10.1.1 (EUF-CMA game)**.** The challenger generates a key pair $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$ and gives the public key pk to $\mathcal{A}$. The adversary $\mathcal{A}$ gets black-box query-access to a signing oracle OSign, which when queried on a message msg outputs a signature $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}, \mathsf{msg})$. The adversary $\mathcal{A}$ wins the game if it can output a message msg^* and a signature σ^* such that $\mathsf{Verify}(\mathsf{pk}, \mathsf{msg}^*, \sigma^*) = 1$ and msg^* has not been queried to OSign.

We define the advantage $\mathsf{Adv}_S^{\mathsf{EUF\text{-}CMA}}(\mathcal{A})$ of $\mathcal{A}$ as the probability that it wins the game. We say that the signature scheme S is EUF-CMA secure if every PPT adversary has negligible advantage in the security parameter λ.

Soundness. SQIsign defines a Σ protocol for the $\mathsf{EndRing}$ relation. It is 2-special sound, but only with respect to the OneEnd problem: given two accepting transcripts for the same commitment, it is possible to extract a non-scalar endomorphism.

Lemma 10.1.2. *If $e_{\mathsf{chl}} + e_{\mathsf{rsp}} \leq e$, then the SQIsign Σ protocol is special-sound for the relation*

$$R_{\mathsf{OneEnd}} = \left\{ (E_{pk}, \alpha) \;\middle|\; \alpha \in \mathrm{End}(E_{pk}) \setminus \mathbb{Z} \text{ in efficient representation with } \deg(\alpha) \leq p^4 \right\}.$$

Zero-knowledge. We now analyze the zero-knowledge property of the SQIsign Σ protocol in the Fiat-Shamir with hints framework. First, we formalize the concept of hints and of a hard relation with hints.

Definition 10.1.3 (Hint distribution)**.** Let $R \subset \mathcal{X} \times \mathcal{W}$ be a relation. A hint distribution $\mathcal{H}$ for R is a collection of distributions $\mathcal{H} = \{\mathcal{H}_x\}_{x \in \mathcal{X}}$, where $\mathcal{H}_x : \mathsf{HintSet}_x \to [0, 1]$ and the elements (i.e., the hints) of $\mathsf{HintSet}_x$ are efficiently representable in $|x|$. The distribution $\mathcal{H}_x$ needs not be efficiently sampleable.

[1] The hints in this framework are additional data that the simulator has access to; they are unrelated to the basis hints that are used in SQIsign to speed up the torsion basis generation, as discussed in Section 2.2.3.

Definition 10.1.4 (Hard relation with hints)**.** Let R be a hard relation and let $\mathcal{H}$ be a hint distribution for R. We call the pair $(R, \mathcal{H})$ a relation with hints. We say that it is a hard relation with hints if, in the following game, it holds for all PPT algorithms $\mathcal{A}$ and all $q = \mathrm{poly}(\lambda)$ that

$$\mathsf{Adv}^{\mathsf{hint\text{-}rel}}_{(R,\mathcal{H})}(\mathcal{A}, q) \coloneqq \Pr\left[(x, w^*) \in R \;\middle|\; \begin{array}{c}(x, w) \leftarrow \mathsf{Gen}_R(1^\lambda),\\ h_1, \ldots, h_q \leftarrow \mathcal{H}_x,\\ w^* \leftarrow \mathcal{A}(x, h_1, \ldots, h_q)\end{array}\right] = \mathsf{negl}(\lambda),$$

where $\mathsf{Gen}_R(1^\lambda)$ is a polynomial-time algorithm that samples instances (x, w) from the distribution R with security parameter 1^λ.

We define a variant of weak honest-verifier zero knowledge (wHVZK) property with hints as follows.

Definition 10.1.5 (Hint-assisted wHVZK)**.** Let Σ be a Σ-protocol for a relation R and let $\mathcal{H}$ be a hint distribution for R. We say that Σ is $\mathcal{H}$-hint-assisted wHVZK if there exists a PPT algorithm, called the simulator $\mathcal{S}$, such that for all $q = \mathrm{poly}(\lambda)$ and all PPT algorithms $\mathcal{A}$

$$\mathsf{Adv}^{\mathsf{hint\text{-}wHVZK}}_{\Sigma,\mathcal{H},\mathcal{S}}(\mathcal{A}, q) \coloneqq \mathsf{Adv}^{\mathsf{dist}}\left[\mathsf{Real}_\Sigma(1^\lambda, q), \mathsf{HintSim}_\Sigma(1^\lambda, q, \mathcal{S})\right](\mathcal{A}) = \mathsf{negl}(\lambda),$$

where $\mathsf{Adv}^{\mathsf{dist}}[\cdot,\cdot](\mathcal{A})$ is the advantage of the adversary $\mathcal{A}$ in distinguishing the two distributions: $\mathsf{Real}_\Sigma(1^\lambda, q)$, which produces q real transcripts of Σ, and $\mathsf{HintSim}_\Sigma(1^\lambda, q, \mathcal{S})$ which outputs q transcripts, simulated by a simulator $\mathcal{S}$ with access to q hints.

In this security proof, we consider two hint distributions, described in Fig. 5. Both distributions sample three isogenies that are used by the simulator to generate a transcript. The first distribution, $\mathcal{H}^{\mathsf{sim}}_E$, samples a challenge, a random commitment curve, and then a connecting isogeny between the challenge curve and the commitment curve. The second distribution, $\mathcal{H}^{\mathsf{unif}}_E$, instead samples a challenge and then an isogeny from the challenge curve to a random curve. This second distribution has the advantage of being "pushable", i.e., given an isogeny from E to E_1 and access to the distribution $\mathcal{H}^{\mathsf{unif}}_E$, it is possible to construct the distribution $\mathcal{H}^{\mathsf{unif}}_{E_1}$. Thus, the simulator would not need $\mathcal{H}^{\mathsf{unif}}_E$ to sample a challenge isogeny, the formulation in Fig. 5 includes it to show the similarity with the $\mathcal{H}^{\mathsf{sim}}_E$ distribution.

$D_{\mathsf{Chall}}(E)$**:**

1: $s \xleftarrow{\$} \{0, \ldots, 2^{e_{\mathsf{chl}}} - 1\}$.
2: $(P, Q), \_ \leftarrow \mathsf{TorsionBasisToHint}(E)$
3: $E', \_ \leftarrow \mathsf{TwoIsogenyChain}(P + [s]Q, f, \_)$ // The 2^f-isogeny with kernel $\langle P + [s]Q\rangle$ has codomain E'
4: return s, E'.

$\mathcal{H}^{\mathsf{sim}}_E$**:**

1: $s, E_1 \leftarrow D_{\mathsf{Chall}}(E)$.
2: Sample an isogeny $\varphi'_2 : E_1 \to E_2$ such that
   i E_2 is uniformly distributed.
   ii The conditional distribution of φ'_2 given E_2 is uniform among isogenies $E_1 \to E_2$ of degree $< 2^{e_{\mathsf{rsp}}}$.
3: Write $\varphi_2 : E_1 \to E_2$ for the cyclic component of φ'_2.
4: Write $\deg(\varphi_2) = 2^n d'$ with d' odd.
5: Sample an isogeny $\varphi_3 : E_2 \to E_3$ uniformly among the cyclic isogenies from E of degree $2^{e_{\mathsf{rsp}}-n} - d'$.
6: return $h = (s, \varphi_2, \varphi_3)$.

$\mathcal{H}^{\mathsf{unif}}_E$**:**

1: $s, E_1 \leftarrow D_{\mathsf{Chall}}(E)$.
2: Sample an integer d from a weighted distribution on the interval $[1, 2^{e_{\mathsf{rsp}}}]$ where each integer n, with prime factorization $n = \prod_{i=1}^t p_i^{e_i}$, has weight $\prod_{i=1}^t (p_i + 1) p_i^{e_i - 1}$.
3: Sample an isogeny $\varphi'_2 : E_1 \to E_2$ uniformly among the (possibly non-cyclic) isogenies from E_1 of degree d.
4: Write $\varphi_2 : E_1 \to E_2$ for the cyclic component of φ'_2.
5: Write $\deg \varphi_2 = 2^n d'$ with d' odd.
6: Sample an isogeny $\varphi_3 : E_2 \to E_3$ uniformly among the cyclic isogenies from E_2 of degree $2^{e_{\mathsf{rsp}}-n} - d'$.
7: return $h = (s, \varphi_2, \varphi_3)$.

FIGURE 5. Hint distributions for SQISIGN.

By relying on the first hint distribution $\mathcal{H}_E^{\mathsf{sim}}$, we can obtain the computational weak honest-verifier zero-knowledge (wHVZK) property of the SQIsign Σ protocol.

Lemma 10.1.6 (Computational hint-assisted wHVZK)**.** *For any $q = \mathrm{poly}(\lambda)$ and any PPT adversary $\mathcal{A}$ against the hint-assisted wHVZK of the SQIsign Σ protocol with q hints, there exists a $\mathcal{H}^{\mathsf{sim}}$-hint-assisted simulator and a PPT algorithm $\mathcal{B}$ such that*

$$\mathsf{Adv}^{\mathsf{hint\text{-}wHVZK}}_{\Sigma_{\mathsf{SQI}},\mathcal{H}^{\mathsf{sim}},}(\mathcal{A},q) \leq \mathsf{Adv}^{\mathsf{EndRing}_p}(\mathcal{B}) + \frac{3(q+1)}{2\sqrt{p}},$$

where $\mathsf{Adv}^{\mathsf{EndRing}_p}(\mathcal{B})$ is the probability that $\mathcal{B}$ solves the EndRing *problem for the prime p and a supersingular curve sampled uniformly at random.*

In the previous lemma, the advantage of $\mathcal{A}$ is bounded by two terms: the first term is the success probability of an $\mathsf{EndRing}_p$ algorithm $\mathcal{B}$ (this is due to the reduction handling cases where any of the curves in the transcript has known endomorphism ring), while the second term is a term that accounts for the statistical distance between the real and simulated transcript distributions.

EUF-CMA security of SQIsign. Finally, we combine the relation R_{OneEnd} with the hint distributions $\mathcal{H}^{\mathsf{sim}}$ to obtain that R_{OneEnd} is a hard relation with $\mathcal{H}^{\mathsf{sim}}$ hints, assuming the hardness of the following problem.

Problem 10.1.7 (q-sim-hint-OneEnd_p)**.** Given a supersingular curve E sampled uniformly at random and q hints $h_1, \ldots, h_q \leftarrow \mathcal{H}_E^{\mathsf{sim}}$, find one non-scalar endomorphism of E in efficient representation.

We can now apply the Fiat-Shamir with hints framework, and we obtain that SQIsign is Existential Unforgeability under Chosen Message Attack (EUF-CMA) secure in the Random Oracle Model (ROM) under the assumption that the above problem is hard.

Theorem 10.1.8 ([ABDF+25, Theorem 2])**.** *Assume that $p \geq 3146$. Then for any PPT algorithm $\mathcal{A}$ against the EUF-CMA of SQIsign, there exists an expected polynomial time algorithm $\mathcal{B}$ such that*

$$\mathsf{Adv}^{\mathsf{EUF\text{-}CMA}}_{SQIsign}(\mathcal{A}) \leq (q+1) \cdot \left(2 \cdot \mathsf{Adv}^{\mathsf{hint\text{-}rel}}_{(R_{\mathsf{OneEnd}},\mathcal{H}^{\mathsf{sim}})}(\mathcal{B},s) + 2^{-e_{\mathsf{chl}}}\right) + (3q+s+3)s \cdot \frac{5}{4\sqrt{p}},$$

where q and s are upper bounds on the number of queries that $\mathcal{A}$ makes to the random oracle and OSign*, respectively.*

Next, we consider using the second hint distribution $\mathcal{H}^{\mathsf{unif}}$. This has several benefits, as we will see. But, it requires us to introduce an additional problem that asks to distinguish between the two distributions.

Problem 10.1.9. Let E be a supersingular curve sampled uniformly at random. Distinguish between the distributions $(E, \mathcal{H}_E^{\mathsf{sim}})$ and $(E, \mathcal{H}_E^{\mathsf{unif}})$.

Then, we consider the relation with hints $(R_{\mathsf{OneEnd}}, \mathcal{H}_E^{\mathsf{unif}})$, whose hardness depends on the following problem.

Problem 10.1.10 (q-unif-hint-OneEnd_p)**.** Given a supersingular curve E sampled uniformly at random and q hints $h_1, \ldots, h_q \leftarrow \mathcal{H}_E^{\mathsf{unif}}$, find one non-scalar endomorphism of E in efficient representation.

Theorem 10.1.11 ([ABDF+25, Theorem 3])**.** *Assume that $p \geq 3146$. Then for any PPT algorithm $\mathcal{A}$ against the EUF-CMA of SQIsign, there exists expected polynomial time algorithms $\mathcal{B}$ and $\mathcal{D}$ such that*

$$\begin{aligned}\mathsf{Adv}^{\mathsf{EUF\text{-}CMA}}_{SQIsign}(\mathcal{A}) \leq{}& (q+1) \cdot \left(2 \cdot \mathsf{Adv}^{\mathsf{hint\text{-}rel}}_{(R_{\mathsf{OneEnd}},\mathcal{H}^{\mathsf{unif}})}(\mathcal{B},s) + 2 \cdot \mathsf{Adv}^{\mathsf{hint\text{-}dist}}(\mathcal{D},s) + 2^{-e_{\mathsf{chl}}}\right) \\ &+ (3qs + s^2 + 3s + 2q + 2) \cdot \frac{5}{4\sqrt{p}},\end{aligned}$$

where q and s are upper bounds on the number of queries that $\mathcal{A}$ makes to the random oracle and OSign*, respectively; the advantages of $\mathcal{B}$ and $\mathcal{D}$ are essentially their success probability for respectively the s-unif-hint-OneEnd_p problem and Problem 10.1.9.*

The advantages of reducing the EUF-CMA security of SQIsign to the q-unif-hint-OneEnd_p problem are twofold. On one hand, the q-unif-hint-OneEnd_p benefits from a worst-case to average-case reduction: this means that as long as there exists a single hard instance of the problem, any average instance of it is similarly hard. Such a reduction is made possible by the fact that the $\mathcal{H}^{\mathsf{unif}}$ hint distribution is pushable, and thus the problem is random self-reducible. On the other hand, the pushability of the $\mathcal{H}^{\mathsf{unif}}$ hint distribution makes it possible to adapt the reduction between the Endomorphism Ring problem and the One Endomorphism problem by Page and

Wesolowski [PW24]. Such an adaption shows that the q-unif-hint-OneEnd$_p$ problem is equivalent, under a tight quantum reduction or a looser classical reduction, to the following generalization of the Endomorphism Ring problem to the hint setting.

Problem 10.1.12 (q-unif-hint-EndRing$_p$)**.** Given a supersingular curve E sampled uniformly at random and q hints $h_1, \ldots, h_q \leftarrow \mathcal{H}_E^{\mathsf{unif}}$, find four endomorphisms in efficient representation that form a basis of $\mathrm{End}(E)$.

The q-unif-hint-EndRing$_p$ is a much more natural problem, and its variant without hints has been extensively studied in the literature. In the following section, we argue why the hints are unlikely to make the problem easier (Section 10.2.3).

10.2. Resistance to known attacks

SQIsign parameters are selected to satisfy the following requirements (among others, see Section 4.2.1):

- The prime $p = c \cdot 2^f - 1$, where c is a small positive integer and $f \approx 2\lambda$.
- The secret and the commitment isogeny are of degree D_{mix}, the smallest prime larger than $2^{4\lambda}$.
- The challenge isogeny degree is $D_{\mathrm{chl}} = 2^f$ with f as above.
- The challenge space has size $2^{e_{\mathrm{chl}}}$ with $e_{\mathrm{chl}} \approx \lambda$.
- The response isogeny degree d_{rsp} is not fixed beforehand, but always satisfies $d_{\mathrm{rsp}} < D_{\mathrm{rsp}} = 2^{e_{\mathrm{rsp}}}$ with $e_{\mathrm{rsp}} = \left\lceil \log_2 \left(\sqrt{p}\right) \right\rceil \approx \lambda$.

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 g, we will write $\widetilde{O}(g)$ to mean $O(g\ell(g))$, 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.

10.2.1. The pure endomorphism ring problem and general isogeny finding

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_{pk}, which can then act as an equivalent secret key.

The best algorithm for finding one non-scalar endomorphism of a random supersingular curve is the algorithm from [EHL+20] with time $O\left((\log p)^2 p^{1/2}\right) \subseteq \widetilde{O}(p^{1/2})$ and low memory requirements. Using a quantum computer and Grover search, the time becomes $\widetilde{O}(p^{1/4})$, still with low memory requirements. The whole endomorphism ring of a supersingular curve can typically be obtained by running the above algorithm $O(1)$ times.

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 it has $\widetilde{O}(p^{1/2})$ classical complexity and $\widetilde{O}(p^{1/4})$ quantum complexity. In SQIsign, the prime p is chosen so that $p \approx 2^{2\lambda}$, so the complexity of the best known attack against the endomorphism ring problem, or equivalently the general isogeny problem, is thus $\widetilde{O}(2^{\lambda})$ classically and $\widetilde{O}(2^{\lambda/2})$ quantumly.

10.2.2. Key recovery

In SQIsign, the secret isogeny $\varphi_{\mathsf{sk}} : E_0 \to E_{\mathsf{pk}}$ is close to a uniformly random isogeny with degree D_{mix}. Alternatively, any (representation of an) isogeny between these two curves could act as an equivalent key, i.e., it is sufficient to sign. The condition $D_{\mathsf{mix}} > 2^{4\lambda}$ guarantees that the distribution of E_{pk} is close to the uniform distribution. By the above equivalence result, computing an isogeny between E_0 and E_{pk} (with a fixed known-endomorphism E_0) is equivalent to the endomorphism ring computation problem, with complexity as stated above.

Given a valid signature, one can also attempt to recover an equivalent secret key in SQIsign by first computing the commitment isogeny between E_0 and E_{com}, or in fact any isogeny between these two curves: given any isogeny between E_0 and the commitment curve, the challenge isogeny, and the signature isogeny, one directly deduces an isogeny from E_0 to E_{pk}. However, the commitment isogeny has the same degree D_{mix} as the secret isogeny: since $D_{\mathsf{mix}} > 2^{4\lambda}$, the distribution of E_{com} is also close to the uniform distribution. Thus, computing an isogeny between E_0 and E_{com} is again equivalent to the endomorphism ring computation problem.

10.2.3. Signature forgery

Forging a signature requires breaking a version of the endomorphism ring problem *with hints*, which a priori could be easier than the generic endomorphism ring problem. In this variant, the adversary is provided with a target curve E and a list of isogenies from E of varying degrees. If the degrees were all smooth, these hints would provide no additional information: anyone can sample random smooth-degree isogenies from any curve, and the two problems would thus be equivalent. In SQIsign, however, the hints contain also isogenies of non-smooth degree, which prevents a reduction between the two problems: this is due to the fact that sampling non-smooth degree isogenies, without additional information on the endomorpshim ring of the starting curve, is generally believed to be hard: a recent isogeny-based proposal [BBC+25] bases its security on such a hardness assumption. Nonetheless, it is unlikely that non-smooth degree isogenies contribute to making the endomorphism ring problem easier: conceptually, non-smooth degree isogenies do not provide any additional information that cannot be already obtained by smooth-degree isogenies. If anything, existing algorithms to compute endomorphisms of a given curve typically use smooth-degree isogenies. Indeed, the best known attack against the endomorphism ring problem with hints disregards the additional information and attempts to compute the endomorphism ring of E directly: thus, the complexity of the best known attack is exactly the same as that for the generic endomorphism problem.

10.2.4. Zero-knowledge

The security of SQIsign relies on a second assumption, the hardness of Problem 10.1.9, which is necessary to guarantee the zero-knowledge property of the underlying sigma protocol. This assumption is a formalization of the heuristic introduced in [BDD+24] to argue the equivalence of the UTO and FIDIO oracles. Our argument for the hardness of Problem 10.1.9 follows a similar approach as [BDD+24].

First, the two distributions in Problem 10.1.9, the $\mathcal{H}^{\mathsf{sim}}$ and the $\mathcal{H}^{\mathsf{unif}}$ distributions, are perfectly the same if conditioned on the codomain curve being the same. If the codomain of φ_3 is fixed, then the two distributions both sample uniformly random connecting isogenies of bounded degree. Thus, any attack against Problem 10.1.9 would need to distinguish between the distributions of the codomain curves: $\mathcal{H}^{\mathsf{sim}}$ samples them uniformly at random, while $\mathcal{H}^{\mathsf{unif}}$ samples them as the codomain of a random isogeny of bounded degree. If this degree were to be sufficiently large (i.e., larger than p^2), the distributions on the codomain curves would be statistically close [DLRW24, Theorem 42], which makes $\mathcal{H}^{\mathsf{sim}}$ and $\mathcal{H}^{\mathsf{unif}}$ statistically indistinguishable. In SQIsign, the degree bound is $O(\sqrt{p})$: thus, we do not achieve statistical indistinguishability, but we expect the two distributions to be computationally indistinguishable.

10.2.5. Attacking the Fiat-Shamir transform

Like any Fiat-Shamir signature, one can also attempt to forge signatures by breaking security properties of the hash function used in the Fiat-Shamir transform (SHAKE). In SQIsign, we use a hash function HASH, with inputs of the form $\mathsf{pk} \mid\mid j(E_{\mathsf{com}}) \mid\mid \mathsf{msg}$, truncated to e_{chl} bits of output. This provides at least e_{chl} bits of classical security against both preimage and second preimage attacks, and $e_{\mathsf{chl}}/2$ bits of resistance against classical collision attacks.

Given a valid signature σ for a message msg, one can use a second preimage attack on the hash function to obtain a second message msg': this leads to the exact same challenge in the SQIsign identification protocol, so that σ is also a valid signature for msg'. This would yield a forgery attack with complexity $O(2^{e_{\mathsf{chl}}})$, which is slightly below the $O(2^\lambda)$ target. To make up for the security gap, we use a hash function that consists of $2^{\lambda - e_{\mathsf{chl}}}$ iterations of a standard hash function, which in our case is SHAKE256. This technique, commonly referred to in the literature as 'grinding', brings the attack cost to the desired $O(2^\lambda)$: finding a collision in HASH requires $O(2^{e_{\mathsf{chl}}})$ attempts, each of which consists of evaluation HASH, which has a cost of at least $2^{\lambda - e_{\mathsf{chl}}}$. Thus, the total attack cost is $O(2^\lambda)$.

It is also possible to consider attacks on the Fiat-Shamir transform. While producing a pair of collisions only costs $2^{e_{\mathsf{chl}}/2}$ bits, this does not seem to lead to more efficient unforgeability attacks on SQIsign. Indeed, the curve E_{com} would a priori need to be fixed before starting the collision search, and the second preimage attack above gives no control on the curve E_{com} to the adversary. A *legitimate signer* could also attempt to produce two messages such that $\mathsf{HASH}(\mathsf{pk} \mid\mid j(E_{\mathsf{com}}) \mid\mid \mathsf{msg}) = \mathsf{HASH}(\mathsf{pk} \mid\mid j(E_{\mathsf{com}}) \mid\mid \mathsf{msg}')$, pretend to sign m, and later deny this by claiming to have signed m'. This attack only requires (fixed prefix) collisions, and hence it can be carried out in time $O(2^{e_{\mathsf{chl}}/2})$. Note that this "deniability attack" goes beyond the basic unforgeability requirement for signatures, and that it also applies to any Fiat-Shamir signature with the same challenge space size.

10.2.6. BUFF security

The Beyond Unforgeability Features (BUFF) [CDF+21] include the security notions of *exclusive ownership* (EO), *message-bound signatures* (MBS) and *non-resignability* (NR). Due to the differences between the round-one SQIsign submission and the present round-two SQIsign submission, the BUFF security analysis for SQIsign from [ADM+24] does not apply.

The BUFF transform [CDF+21] reduces the security of a signature scheme with regard to all BUFF notions to the collision resistance of the underlying hash function by signing a hash of the message and public key. Since in SQIsign the challenge $\mathsf{chl} = \mathsf{HASH}(\mathsf{pk} \mid\mid j(E_{\mathsf{com}}) \mid\mid \mathsf{msg})$ of length $\approx \lambda$ is part of the signature, and takes the public key as an input, we use an implicit form of the BUFF transform by design. Hence, SQIsign achieves at least $\approx \lambda/2$ bits of security for all BUFF notions, while this bound is only tight for MBS.

SQIsign can easily be adapted to achieve λ bits of BUFF security at the cost of a minor increase in signature size: increasing the hash output length to get $\mathsf{chl}_{2\lambda} = \mathsf{HASH}(\mathsf{pk} \mid\mid j(E_{\mathsf{com}}) \mid\mid \mathsf{msg})$ of bit length 2λ as part of the signature ensures sufficient collision resistance. During signing and verifying, only the first $\approx \lambda$ bits of $\mathsf{chl}_{2\lambda}$ are used as chl, and we follow the exact same signing and verifying procedure, featuring a challenge space of $\approx \lambda$ bit, as above. This modification is similar to the way ML-DSA achieves BUFF security, and increases the signature sizes in SQIsign by $\lambda/8$ bytes due to the larger size of $\mathsf{chl}_{2\lambda}$.

10.2.7. Strong unforgeability

Note that SQIsign does not target *strong* unforgeability security, and indeed given a valid signature on a message, one can efficiently produce a second distinct valid signature on the same message by manipulating the auxiliary isogeny. Replacing the auxiliary isogeny with any other isogeny of the same degree yields a valid signature.

The role of the auxiliary isogeny in the signature is only to enable a two-dimensional representation of the response, but it does not contribute to the security of the protocol. In other words, two-dimensional representations are inherently *not* unique: given such a representation, in most instances it is easy to find a different representation of the same isogeny. For this reason, SQIsign cannot achieve strong unforgeability.

10.2.8. Algorithm failures and their impact on security

The implementation of SQIsign relies on some algorithms that can possibly fail rather than producing the desired output, as discussed in Chapter 9. We identify three such high-level algorithms: the computation of $(2^n, 2^n)$-isogenies, the translation of ideals to their corresponding isogeny, and the sampling of the auxiliary isogeny. The protocol handles these cases differently depending on where the failure occurs: if any subroutine fails during key generation or signing, the protocol restarts the entire operation with different randomness; during verification, an algorithmic failure leads to a signature rejection.

These failure cases always occur with small or negligible probability, so they have a limited impact on the security of SQIsign. We now analyze the three algorithms separately, discussing their precise impact on the protocol.

$(2^n, 2^n)$-isogeny computation. The computation of $(2^n, 2^n)$-isogenies occurs in multiple places in SQIsign: within the translation of ideals to isogenies (during signing) and in the computation of the isogeny Φ (during signing and verification). However, in specific circumstances, the algorithm described in Section 2.4.1 may fail. When a failure occurs during signing, the signer restarts the procedure; when the failure occurs during verification, the verifier outputs `false` since a failure indicates with overwhelming probability (roughly, $1 - 2^{-4\lambda}$) a malicious signature.

The computation of $(2^n, 2^n)$-isogenies may fail for multiple reasons, including the presence of products of elliptic curves in the middle of the chain. However, as argued in Chapter 9, these failures occur with negligible probability $O(2^{-2\lambda})$. As such, they do not impact the security of SQIsign.

Ideal to isogeny translation. The algorithm to translate ideals into isogenies relies on several algorithms that may fail: these include the computation of $(2^n, 2^n)$-isogenies (discussed above), sampling endomorphisms of a given degree, and finding equivalent ideals to the input ideal that satisfy certain conditions (which is necessary to translate the ideal using only isogenies in dimension two). While the failure probability of each of these subroutines varies considerably, the protocol parameters are chosen to guarantee that the ideal to isogeny translation only fails with a probability lower than 2^{-64}, for all three security levels.

Let us consider the impact of such failure cases on security separately, depending on when they occur:

- **Key Generation:** the secret ideal is translated into its corresponding isogeny to obtain the public key curve E_{pk}. If a failure occurs, the key generation restarts with different randomness. The majority of failure cases depend on the equivalence class of the input ideal: this means that whether the algorithm fails will depend, with good probability, on the public-key curve E_{pk}. Hence, some curves may never be generated by the key generation procedure, which shows that the failure cases bias the distribution of public keys. However, this does not affect security since the biased distribution is very close to the uniform distribution (the statistical distance between the two is less than 2^{-64}). This means that any adversary against the biased distribution can be turned into an adversary against the uniform distribution (see [ABDF+25] for a formal proof).
- **Commitment generation:** similarly to the case of key generation, SQIsign generates a random ideal and translates into its corresponding isogeny to sample a commitment curve. If a failure occurs, the signing procedure samples a different commitment ideal: thus, the failure cases in the ideal-to-isogeny translation algorithm bias the distribution of the commitment curve. While this does not affect the hardness of the endomorphism problem or the soundness of the protocol, it may have an impact on the zero-knowledge property. However, we do not expect it to be the case because any realistic attack scenario would limit the number of signatures accessible to the adversary to, at most, 2^{64} signatures per public key.[2] Thus, the failure probability is exactly the inverse of the maximum number of signatures: even an attacker with access to such a large number of signatures is expected to see only one curve that is the result of a failure in the ideal-to-isogeny algorithm. This curve is sampled from an exponentially large subset of all possible curves; it is thus unlikely that seeing such a curve helps the attacker to distinguish the two distributions. Moreover, the failure probability depends on the ideal class connecting E_0 to the commitment curve: we thus expect that to even detect a bias, the adversary would need non-trivial knowledge on the connecting ideal, and therefore non-trivial knowledge on the endomorphism ring of the commitment curve.
- **Auxiliary isogeny translation:** the ideal-to-isogeny translation algorithm is also a subroutine in the sampling of the auxiliary isogeny, which is part of the response generation. We treat this separately since the sampling of the auxiliary isogeny may fail also for other reasons.

Auxiliary isogeny sampling. The last routine that may fail is the sampling of the auxiliary isogeny. This may happen for two reasons: the algorithm that generates a random ideal of composite norm may fail, or the ideal-to-isogeny translation may fail. If a failure occurs, the entire signing procedure restarts. The analysis of this case is similar to that of commitment generations: the low failure rate and the limited number of signatures available to an adversary makes it unlikely for an attacker to distinguish a real transcript from a simulated one.

10.3. Nothing up my sleeve

Here we provide some additional comments on our parameter choices, arguing that these have 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 it is a standard choice in the literature to ensure that 1728 is the j-invariant of a supersingular elliptic curve. This choice reduces the set of potential parameters by only half, and we are not aware of any impact on the computational assumptions underlying SQIsign.
- More specifically, the primes we use are of a very specific shape, namely $p = c \cdot 2^f - 1$, where c is a small positive integer. The special shape is motivated by efficiency considerations: it enables the efficient use of $(2, 2)$-isogenies instead of the (d, d)-isogenies, for various small d, that would be required a priori by generic primes. In addition, the special prime shape supports very efficient arithmetic over $\mathbb{F}_p$ and $\mathbb{F}_{p^2}$. Overall, these factors have a large performance impact. On the other hand, there is no evidence that this prime selection affects the hardness of the endomorphism ring computation problem beyond the polynomial speedups gained from faster arithmetic over $\mathbb{F}_{p^2}$. We also note that, similarly, SIKE employed special primes (of the form $p = c \cdot 2^f 3^e - 1$) with no known impact on security (the SIKE attacks work independently of the shape of p).

[2] The NIST call for additional digital signature schemes explicitly mentions that *[f]or the purpose of estimating security strengths, it may be assumed that the attacker has access to signatures for no more than* 2^{64} *chosen messages*. Accessed at `https://csrc.nist.gov/csrc/media/Projects/pqc-dig-sig/documents/call-for-proposals-dig-sig-sept-2022.pdf` on February 2nd, 2025.

- As is typically the case in isogeny-based cryptography, the curve E_0 with j-invariant $j = 1728$ plays a special role in the algorithms. However, in contrast to other schemes (including the Round 1 version of SQIsign), this choice is provably independent of security. The critical elliptic curves E_{pk} and E_{com} are uniformly distributed, with no correlation with E_0. This uniform distribution leverages the worst-case to average-case reduction of the endomorphism ring problem, making the scheme's security dependent on the problem's worst-case hardness.

Bibliography

[AAA+25] Marius A. Aardal, Gora Adj, Arwa Alblooshi, Diego F. Aranha, Isaac Andrés Canales Martinez, Jorge Chávez-Saab, Décio Luiz Gazzoni Filho, Krijn Reijnders, and Francisco Rodríguez-Henríquez. "Optimized One-Dimensional SQIsign Verification on Intel and Cortex-M4". In: *IACR TCHES* 2025.1 (2025), pp. 497–522. DOI: 10.46586/tches.v2025.i1.497-522 (cit. on p. 10).

[ABDF+25] Marius A. Aardal, Andrea Basso, Luca De Feo, Sikhar Patranabis, and Benjamin Wesolowski. *A detailed security proof of SQIsign*. https://sqisign.org/link/securityproof. 2025 (cit. on pp. 89, 91, 95).

[ADM+24] Thomas Aulbach, Samed Düzlü, Michael Meyer, Patrick Struck, and Maximiliane Weishäupl. "Hash Your Keys Before Signing - BUFF Security of the Additional NIST PQC Signatures". In: *Post-Quantum Cryptography - 15th International Workshop, PQCrypto 2024, Part II*. Ed. by Markku-Juhani Saarinen and Daniel Smith-Tone. Springer, Cham, June 2024, pp. 301–335. DOI: 10.1007/978-3-031-62746-0_13 (cit. on p. 94).

[Apo76] Tom M. Apostol. *Introduction to Analytic Number Theory*. Springer New York, 1976. ISBN: 9781475755794. DOI: 10.1007/978-1-4757-5579-4 (cit. on p. 87).

[BBC+25] Andrea Basso, Giacomo Borin, Wouter Castryck, Maria Corte-Real Santos, Riccardo Invernizzi, Antonin Leroux, Luciano Maino, Frederik Vercauteren, and Benjamin Wesolowski. *PRISM: Simple And Compact Identification and Signatures From Large Prime Degree Isogenies*. Cryptology ePrint Archive, Paper 2025/135. 2025. URL: https://eprint.iacr.org/2025/135 (cit. on p. 93).

[BDD+24] Andrea Basso, Pierrick Dartois, Luca De Feo, Antonin Leroux, Luciano Maino, Giacomo Pope, Damien Robert, and Benjamin Wesolowski. "SQIsign2D-West - The Fast, the Small, and the Safer". In: *ASIACRYPT 2024, Part III*. Ed. by Kai-Min Chung and Yu Sasaki. Vol. 15486. LNCS. Springer, Singapore, Dec. 2024, pp. 339–370. DOI: 10.1007/978-981-96-0891-1_11 (cit. on pp. 6, 93).

[BKLS02] Paulo S. L. M. Barreto, Hae Yong Kim, Ben Lynn, and Michael Scott. "Efficient Algorithms for Pairing-Based Cryptosystems". In: *CRYPTO 2002*. Ed. by Moti Yung. Vol. 2442. LNCS. Springer, Berlin, Heidelberg, Aug. 2002, pp. 354–368. DOI: 10.1007/3-540-45708-9_23 (cit. on p. 62).

[Can87] David Geoffrey Cantor. "Computing in the Jacobian of a hyperelliptic curve". In: *Mathematics of Computation* 48 (1987), pp. 95–101. URL: https://api.semanticscholar.org/CorpusID:121203451 (cit. on p. 18).

[Cas78] John W. S. Cassels. *Rational Quadratic Forms*. Dover books on mathematics. Academic Press, 1978. ISBN: 9780121632601. URL: https://books.google.ch/books?id=RnOpAQAAMAAJ (cit. on p. 87).

[CD23] Wouter Castryck and Thomas Decru. "An Efficient Key Recovery Attack on SIDH". In: *EUROCRYPT 2023, Part V*. Ed. by Carmit Hazay and Martijn Stam. Vol. 14008. LNCS. Springer, Cham, Apr. 2023, pp. 423–447. DOI: 10.1007/978-3-031-30589-4_15 (cit. on p. 4).

[CDF+21] Cas Cremers, Samed Düzlü, Rune Fiedler, Marc Fischlin, and Christian Janson. "BUFFing signature schemes beyond unforgeability and the case of post-quantum signatures". In: *2021 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, May 2021, pp. 1696–1714. DOI: 10.1109/SP40001.2021.00093 (cit. on p. 94).

[CDLR25] Maria Corte-Real Santos, Max Duparc, Antonin Leroux, and Damien Robert. *A detailed explanation of the computation of* $(2,2)$*-isogenies in SQIsign*. https://sqisign.org/link/22isogenies. 2025 (cit. on p. 69).

[CJL+17] Craig Costello, David Jao, Patrick Longa, Michael Naehrig, Joost Renes, and David Urbanik. "Efficient Compression of SIDH Public Keys". In: *EUROCRYPT 2017, Part I*. Ed. by Jean-Sébastien Coron and Jesper Buus Nielsen. Vol. 10210. LNCS. Springer, Cham, 2017, pp. 679–706. DOI: 10.1007/978-3-319-56620-7_24 (cit. on p. 12).

[CLN16] Craig Costello, Patrick Longa, and Michael Naehrig. "Efficient Algorithms for Supersingular Isogeny Diffie-Hellman". In: *CRYPTO 2016, Part I*. Ed. by Matthew Robshaw and Jonathan Katz. Vol. 9814. LNCS. Springer, Berlin, Heidelberg, Aug. 2016, pp. 572–601. DOI: 10.1007/978-3-662-53018-4_21 (cit. on p. 17).

[Coh93] Henri Cohen. *A Course in Computational Algebraic Number Theory*. New York, NY, USA: Springer, 1993. ISBN: 978-3-540-55640-4. DOI: 10.1007/978-3-662-02945-9 (cit. on pp. 23, 27).

[Cor08] Giuseppe Cornacchia. "Su di un metodo per la risoluzione in numeri interi dell'equazione $\sum_{h=0}^{n} C_h x^{n-h} y^h = P$". In: *Giornale di Matematiche di Battaglini* 46 (1908), pp. 33–90 (cit. on p. 30).

[CS18] Craig Costello and Benjamin Smith. "Montgomery curves and their arithmetic - The case of large characteristic fields". In: *Journal of Cryptographic Engineering* 8.3 (Sept. 2018), pp. 227–240. DOI: 10.1007/s13389-017-0157-6 (cit. on pp. 10, 11).

[Dar24] Pierrick Dartois. *Fast computation of 2-isogenies in dimension 4 and cryptographic applications*. Cryptology ePrint Archive, Report 2024/1180. 2024. URL: https://eprint.iacr.org/2024/1180 (cit. on p. 70).

[Die75] U. Dieter. "How to calculate shortest vectors in a lattice". In: *Mathematics of Computation* 29.131 (1975), 827–833. ISSN: 1088-6842. DOI: 10.1090/s0025-5718-1975-0379386-6 (cit. on p. 26).

[DKL+20] Luca De Feo, David Kohel, Antonin Leroux, Christophe Petit, and Benjamin Wesolowski. "SQISign: Compact Post-quantum Signatures from Quaternions and Isogenies". In: *ASIACRYPT 2020, Part I*. Ed. by Shiho Moriai and Huaxiong Wang. Vol. 12491. LNCS. Springer, Cham, Dec. 2020, pp. 64–93. DOI: 10.1007/978-3-030-64837-4_3 (cit. on pp. 4, 28).

[DLLW23] Luca De Feo, Antonin Leroux, Patrick Longa, and Benjamin Wesolowski. "New Algorithms for the Deuring Correspondence - Towards Practical and Secure SQISign Signatures". In: *EUROCRYPT 2023, Part V*. Ed. by Carmit Hazay and Martijn Stam. Vol. 14008. LNCS. Springer, Cham, Apr. 2023, pp. 659–690. DOI: 10.1007/978-3-031-30589-4_23 (cit. on p. 31).

[DLRW24] Pierrick Dartois, Antonin Leroux, Damien Robert, and Benjamin Wesolowski. "SQIsignHD: New Dimensions in Cryptography". In: *EUROCRYPT 2024, Part I*. Ed. by Marc Joye and Gregor Leander. Vol. 14651. LNCS. Springer, Cham, May 2024, pp. 3–32. DOI: 10.1007/978-3-031-58716-0_1 (cit. on pp. 86, 93).

[DMPR24] Pierrick Dartois, Luciano Maino, Giacomo Pope, and Damien Robert. "An Algorithmic Approach to (2, 2)-Isogenies in the Theta Model and Applications to Isogeny-Based Cryptography". In: *ASIACRYPT 2024, Part III*. Ed. by Kai-Min Chung and Yu Sasaki. Vol. 15486. LNCS. Springer, Singapore, Dec. 2024, pp. 304–338. DOI: 10.1007/978-981-96-0891-1_10 (cit. on pp. 69–71, 74).

[Ebe08] David Eberly. *The Laplace Expansion Theorem: Computing the Determinants and Inverses of Matrices*. 2008. URL: https://www.geometrictools.com/Documentation/LaplaceExpansionTheorem.pdf (cit. on pp. 23, 27).

[EHL+18] Kirsten Eisenträger, Sean Hallgren, Kristin E. Lauter, Travis Morrison, and Christophe Petit. "Supersingular Isogeny Graphs and Endomorphism Rings: Reductions and Solutions". In: *EUROCRYPT 2018, Part III*. Ed. by Jesper Buus Nielsen and Vincent Rijmen. Vol. 10822. LNCS. Springer, Cham, 2018, pp. 329–368. DOI: 10.1007/978-3-319-78372-7_11 (cit. on pp. 5, 92).

[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". In: *Open Book Series* 4.1 (2020), pp. 215–232. DOI: 10.2140/obs.2020.4.215 (cit. on p. 92).

[ESPV23] Jonathan Komada Eriksen, Jana Sotáková, Lorenz Panny, and Mattia Veroni. "Deuring for the people: Supersingular elliptic curves with prescribed endomorphism ring in general characteristic". In: *LuCaNT 2023*. 2023. DOI: https://doi.org/10.1090/conm/796/16008 (cit. on p. 33).

[FHLOJRH18] Armando Faz-Hernández, Julio López, Eduardo Ochoa-Jiménez, and Francisco Rodríguez-Henríquez. "A Faster Software Implementation of the Supersingular Isogeny Diffie-Hellman Key Exchange Protocol". In: *IEEE Transactions on Computers* 67.11 (2018), pp. 1622–1636. DOI: 10.1109/TC.2017.2771535 (cit. on p. 53).

[FP85] Ulrich Fincke and Michael Pohst. "Improved methods for calculating vectors of short length in a lattice, including a complexity analysis". In: *Mathematics of Computation* 44.170 (1985), pp. 463–471. DOI: 10.1090/S0025-5718-1985-0777278-8 (cit. on p. 26).

[FPLLL] The FPLLL development team. "fplll, a lattice reduction library, Version: 5.5.0". Available at https://github.com/fplll/fplll. 2023. URL: https://github.com/fplll/fplll (cit. on p. 26).

[FR94] Gerhard Frey and Hans-Georg Rück. "A remark concerning m-divisibility and the discrete logarithm in the divisor class group of curves". In: *Mathematics of computation* 62.206 (1994), pp. 865–874. DOI: https://doi.org/10.2307/2153546 (cit. on p. 15).

[FS87] Amos Fiat and Adi Shamir. "How to Prove Yourself: Practical Solutions to Identification and Signature Problems". In: *CRYPTO'86*. Ed. by Andrew M. Odlyzko. Vol. 263. LNCS. Springer, Berlin, Heidelberg, Aug. 1987, pp. 186–194. DOI: 10.1007/3-540-47721-7_12 (cit. on p. 37).

[FST10] David Freeman, Michael Scott, and Edlyn Teske. "A Taxonomy of Pairing-Friendly Elliptic Curves". In: *Journal of Cryptology* 23.2 (Apr. 2010), pp. 224–280. DOI: 10.1007/s00145-009-9048-z (cit. on p. 62).

[Ibu82] Tomoyoshi Ibukiyama. "On maximal orders of division quaternion algebras over the rational number field with certain optimal embeddings". In: *Nagoya Mathematical Journal* 88 (1982), pp. 181–195. DOI: 10.1017/S002776300002016X (cit. on p. 33).

[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*. Tech. rep. available at https://csrc.nist.gov/projects/post-quantum-cryptography/post-quantum-cryptography-standardization/round-3-submissions. National Institute of Standards and Technology, 2020 (cit. on pp. 9, 16, 17).

[JD11] David Jao and Luca De Feo. "Towards Quantum-Resistant Cryptosystems from Supersingular Elliptic Curve Isogenies". In: *Post-Quantum Cryptography - 4th International Workshop, PQCrypto 2011*. Ed. by Bo-Yin Yang. Springer, Berlin, Heidelberg, 2011, pp. 19–34. DOI: 10.1007/978-3-642-25405-5_2 (cit. on pp. 4, 19).

[Koh96] David Kohel. "Endomorphism rings of elliptic curves over finite fields". PhD thesis. University of California, Berkeley, 1996. ISBN: 978-0591-32123-4 (cit. on p. 5).

[KPR+] Matthias J. Kannwischer, Richard Petri, Joost Rijneveld, Peter Schwabe, and Ko Stoffelen. *PQM4: Post-quantum crypto library for the ARM Cortex-M4*. https://github.com/mupq/pqm4 (cit. on p. 51).

[LB20] Jonathan Love and Dan Boneh. "Supersingular curves with small noninteger endomorphisms". In: *Open Book Series* 4.1 (2020), pp. 7–22. DOI: 10.2140/obs.2020.4.7 (cit. on p. 86).

[Lic69] Stephen Lichtenbaum. "Duality theorems for curves over p-adic fields". In: *Inventiones mathematicae* 7.2 (1969), pp. 120–136. DOI: https://doi.org/10.1007/BF01389795 (cit. on p. 15).

[Lon23] Patrick Longa. "Efficient Algorithms for Large Prime Characteristic Fields and Their Application to Bilinear Pairings". In: *IACR TCHES* 2023.3 (2023), pp. 445–472. DOI: 10.46586/tches.v2023.i3.445-472 (cit. on pp. 53, 54).

[LR22] David Lubicz and Damien Robert. "Fast change of level and applications to isogenies". In: *Research in Number Theory* 9.1 (2022), p. 7. DOI: 10.1007/s40993-022-00407-9 (cit. on p. 18).

[LWXZ24] Kaizhan Lin, Weize Wang, Zheng Xu, and Chang-An Zhao. "A Faster Software Implementation of SQIsign". In: *IEEE Trans. Inf. Theory* 70.9 (2024), pp. 6679–6689. DOI: 10.1109/TIT.2024.3423675 (cit. on p. 14).

[Mil04] Victor S. Miller. "The Weil Pairing, and Its Efficient Calculation". In: *Journal of Cryptology* 17.4 (Sept. 2004), pp. 235–261. DOI: 10.1007/s00145-004-0315-8 (cit. on pp. 62, 63).

[Mil86] J. S. Milne. "Jacobian Varieties". In: *Arithmetic Geometry*. Ed. by Gary Cornell and Joseph H. Silverman. New York, NY: Springer New York, 1986, pp. 167–212. DOI: 10.1007/978-1-4613-8655-1_7 (cit. on p. 17).

[MMP+23] Luciano Maino, Chloe Martindale, Lorenz Panny, Giacomo Pope, and Benjamin Wesolowski. "A Direct Key Recovery Attack on SIDH". In: *EUROCRYPT 2023, Part V*. Ed. by Carmit Hazay and Martijn Stam. Vol. 14008. LNCS. Springer, Cham, Apr. 2023, pp. 448–471. DOI: 10.1007/978-3-031-30589-4_16 (cit. on p. 4).

[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 (cit. on p. 30).

[NS09] Phong Q. Nguyen and Damien Stehlé. "An LLL Algorithm with Quadratic Complexity". In: *SIAM Journal on Computing* 39.3 (2009), pp. 874–903. DOI: 10.1137/070705702. eprint: https://doi.org/10.1137/070705702. URL: https://doi.org/10.1137/070705702 (cit. on pp. 24, 26).

[NV10] Phong Q. Nguyen and Brigitte Vallée, eds. *The LLL Algorithm - Survey and Applications*. myisc. Springer, 2010. ISBN: 978-3-642-02294-4. DOI: 10.1007/978-3-642-02295-1 (cit. on p. 24).

[OU73] Frans Oort and Kenji Ueno. "Principally polarized abelian variaties of dimension two or three are Jacobian varieties". Feb. 1973 (cit. on p. 17).

[PH78] Stephen Pohlig and Martin Hellman. "An Improved Algorithm for Computing Logarithms over $\mathrm{GF}(p)$ and Its Cryptographic Significance". In: *IEEE Transactions on Information Theory* 24.1 (Jan. 1978), pp. 106–110. DOI: 10.1109/TIT.1978.1055817 (cit. on p. 13).

[Por20a] Thomas Pornin. *Faster Modular Inversion and Legendre Symbol, and an X25519 Speed Record*. https://research.nccgroup.com/2020/09/28/faster-modular-inversion-and-legendre-symbol-and-an-x25519-speed-record/. 2020 (cit. on p. 54).

[Por20b] Thomas Pornin. *Optimized Binary GCD for Modular Inversion*. Cryptology ePrint Archive, Paper 2020/972. 2020. URL: https://eprint.iacr.org/2020/972 (cit. on p. 54).

[PW24] Aurel Page and Benjamin Wesolowski. "The Supersingular Endomorphism Ring and One Endomorphism Problems are Equivalent". In: *EUROCRYPT 2024, Part VI*. Ed. by Marc Joye and Gregor Leander. Vol. 14656. LNCS. Springer, Cham, May 2024, pp. 388–417. DOI: 10.1007/978-3-031-58751-1_14 (cit. on pp. 6, 92).

[PZ24] Patrick Pelissier and Paul Zimmerman. *The DPE library*. 2024. URL: https://gitlab.inria.fr/zimmerma/dpe/ (cit. on p. 22).

[Rei23] Krijn Reijnders. "Effective Pairings in Isogeny-Based Cryptography". In: *LATINCRYPT 2023*. Ed. by Abdelrahaman Aly and Mehdi Tibouchi. Vol. 14168. LNCS. Springer, Cham, Oct. 2023, pp. 109–128. DOI: 10.1007/978-3-031-44469-2_6 (cit. on p. 62).

[Ren18] Joost Renes. "Computing Isogenies Between Montgomery Curves Using the Action of (0, 0)". In: *Post-Quantum Cryptography - 9th International Conference, PQCrypto 2018*. Ed. by Tanja Lange and Rainer Steinwandt. Springer, Cham, 2018, pp. 229–247. DOI: 10.1007/978-3-319-79063-3_11 (cit. on p. 16).

[Rob23] Damien Robert. "Breaking SIDH in Polynomial Time". In: *EUROCRYPT 2023, Part V*. Ed. by Carmit Hazay and Martijn Stam. Vol. 14008. LNCS. Springer, Cham, Apr. 2023, pp. 472–503. DOI: 10.1007/978-3-031-30589-4_17 (cit. on p. 4).

[Rob24] Damien Robert. *Fast pairings via biextensions and cubical arithmetic*. Cryptology ePrint Archive, Report 2024/517. 2024. URL: https://eprint.iacr.org/2024/517 (cit. on pp. 62, 63).

[RS17] Joost Renes and Benjamin Smith. "qDSA: Small and Secure Digital Signatures with Curve-Based Diffie-Hellman Key Pairs". In: *ASIACRYPT 2017, Part II*. Ed. by Tsuyoshi Takagi and Thomas Peyrin. Vol. 10625. LNCS. Springer, Cham, Dec. 2017, pp. 273–302. DOI: 10.1007/978-3-319-70697-9_10 (cit. on p. 12).

[Sco24] Michael Scott. *Elliptic Curve Cryptography for the masses: Simple and fast finite field arithmetic*. Cryptology ePrint Archive, Report 2024/779. 2024. URL: https://eprint.iacr.org/2024/779 (cit. on p. 53).

[SEMR24] Maria Corte-Real Santos, Jonathan Komada Eriksen, Michael Meyer, and Krijn Reijnders. "AprèsSQI: Extra Fast Verification for SQIsign Using Extension-Field Signing". In: *EUROCRYPT 2024, Part I*. Ed. by Marc Joye and Gregor Leander. Vol. 14651. LNCS. Springer, Cham, May 2024, pp. 63–93. DOI: 10.1007/978-3-031-58716-0_3 (cit. on p. 12).

[Sta07] Katherine E. Stange. "The Tate Pairing Via Elliptic Nets". In: *PAIRING 2007*. Ed. by Tsuyoshi Takagi, Tatsuaki Okamoto, Eiji Okamoto, and Takeshi Okamoto. Vol. 4575. LNCS. Springer,

Berlin, Heidelberg, July 2007, pp. 329–348. DOI: 10.1007/978-3-540-73489-5_19 (cit. on p. 62).

[Sta08] Katherine Stange. "Elliptic nets and elliptic curves". PhD thesis. Brown University, 2008. URL: https://repository.library.brown.edu/studio/item/bdr:309/PDF/ (cit. on p. 63).

[Tat62] John Tate. "Duality theorems in Galois cohomology over number fields". In: *Proc. Internat. Congr. Mathematicians (Stockholm, 1962)*. 1962, pp. 288–295 (cit. on p. 15).

[Vél71] Jacques Vélu. "Isogénies entre courbes elliptiques". In: *Comptes Rendus de l'Académie des Sciences de Paris, Séries A* 273 (1971), pp. 238–241 (cit. on p. 16).

[Wei40] André Weil. "Sur les fonctions algébriques a corps de constantes fini". In: *CR Acad. Sci. Paris* 210.1940 (1940), pp. 592–594 (cit. on p. 15).

[Wes22] Benjamin Wesolowski. "The supersingular isogeny path and endomorphism ring problems are equivalent". In: *62nd FOCS*. IEEE Computer Society Press, Feb. 2022, pp. 1100–1111. DOI: 10.1109/FOCS52979.2021.00109 (cit. on pp. 5, 85, 92).

[ZSP+18] Gustavo HM Zanon, Marcos A Simplicio, Geovandro CCF Pereira, Javad Doliskani, and Paulo SLM Barreto. "Faster key compression for isogeny-based cryptosystems". In: *IEEE Transactions on Computers* 68.5 (2018), pp. 688–701. DOI: 10.1109/TC.2018.2878829 (cit. on p. 12).

APPENDIX A

Lemmas on lattice sampling

We prove here that Algorithm LatticeSampling on page 26 is correct and we give a heuristic estimate for its running time. In what follows $\langle \mathbf{x}, \mathbf{x} \rangle$ denotes a positive definite symmetric bilinear form and $Q(\mathbf{x}) = \langle \mathbf{x}, \mathbf{x} \rangle$ its associated quadratic form.

Lemma A.1. *Let $Q(\mathbf{x})$ be a positive definite quadratic form on $\mathbb{R}^d$ represented by a Gram matrix $\mathbf{G}$. Let $\mathbf{G}^{-1} = (g'_{i,j})$ and $\mathbf{x} = (x_i)^t$, then*

$$x_i^2 \leq g'_{i,i} Q(\mathbf{x}) = g'_{i,i} \mathbf{x}^t \mathbf{G} \mathbf{x}.$$

Proof. Write $(\mathbf{e}_i)$ for the canonical basis of $\mathbb{R}^d$, so that $\langle \mathbf{e}_i, \mathbf{e}_j \rangle = \mathbf{e}_i^t \mathbf{G} \mathbf{e}_j = g_{i,j}$. Let $(\mathbf{e}_i^\star)$ be the basis of the dual lattice to $(\mathbf{e}_i)$ such that $\mathbf{e}_i^\star(\mathbf{e}_j) = \delta_{i,j}$, it has an associated quadratic form $Q(\mathbf{x}^\star)^\star = \mathbf{x}^\star \mathbf{G}^{-1} (\mathbf{x}^\star)^t$. To conclude observe that

$$x_i^2 = \mathbf{e}_i^\star(\mathbf{x})^2 \leq Q(\mathbf{e}_i)^\star \, Q(\mathbf{x}) = g'_{i,i} \mathbf{x}^t \mathbf{G} \mathbf{x},$$

where the inequality follows from the Cauchy–Schwarz inequality. □

Corollary A.2. *Algorithm LatticeSampling on page 26 is correct: the vector it outputs is distributed uniformly among those of norm $\leq B$.*

Proof. Let Λ be the lattice generated by $(\mathbf{b}_0, \ldots, \mathbf{b}_{d-1})$ and let $(\mathbf{b}_0^\star, \ldots, \mathbf{b}_{d-1}^\star)$ be the basis of the dual lattice $\Lambda^\star$. After the call to L2 the bases $(\mathbf{b}_0^\star \cdots \mathbf{b}_{d-1}^\star)\mathbf{U}$ and $\mathbf{U}^{-1}(\mathbf{b}_0 \cdots \mathbf{b}_{d-1})^t$, generating $\Lambda^\star$ and Λ respectively, are dual to one another; $\mathbf{H}$ is the Gram matrix of the former, and $\mathbf{G}' = \mathbf{U}^{-1}\mathbf{G}(\mathbf{U}^{-1})^t$ of the latter.

Write $\mathbf{x} = (x_0 \cdots x_{d-1})$ and $\mathbf{y} = (y_0 \cdots y_{d-1})$, assume that

$$\mathbf{y}\mathbf{G}\mathbf{y}^t = \mathbf{x}\mathbf{G}'\mathbf{x}^t \leq B,$$

then by Lemma A.1

$$x_i^2 \leq \mathbf{H}_{i,i} \mathbf{x} \mathbf{G}' \mathbf{x}^t \leq \mathbf{H}_{i,i} B.$$

Thus, rejection sampling from the parallelogram with bounds $|x_i| \leq \lfloor \sqrt{\mathbf{H}_{i,i} B} \rfloor$ ensures that every lattice vector of length $\leq B$ is selected with the same probability. □

Getting a precise bound for the rejection probability of the algorithm seems difficult, however we can heuristically approximate it by bounding the ratio between the volume of the parallelogram where the algorithm samples and the volume of the ellipsoid that contains all (non-lattice) vectors of length $\leq B$.

Lemma A.3. *Let $\mathbf{G}$ be an (η, δ)-reduced positive definite Gram matrix. Define $\theta = (\delta - \eta^2)^{-1}$ then*

$$\frac{(\theta - 1)^d}{\theta^{d(d-1)/2}} \prod \mathbf{G}_{i,i} \leq \det(\mathbf{G}) \leq \prod \mathbf{G}_{i,i}.$$

Proof. Let $(\mathbf{b}_0, \ldots, \mathbf{b}_1)$ be an (η, δ)-reduced basis of a quadratic space such that $\langle \mathbf{b}_i, \mathbf{b}_j \rangle = \mathbf{G}_{i,j}$. Recall from Definition 3.1.1 the definition of the Gram-Schmidt vectors:

$$\overline{\mathbf{b}}_i = \mathbf{b}_i - \sum_{j=0}^{i-1} \mu_{i,j} \overline{\mathbf{b}}_j \quad \text{where} \quad \left\langle \mathbf{b}_i, \overline{\mathbf{b}}_j \right\rangle = \mu_{i,j} \left\langle \overline{\mathbf{b}}_j, \overline{\mathbf{b}}_j \right\rangle.$$

The $\overline{\mathbf{b}}_i$ are orthogonal and related to the $\mathbf{b}_i$ by a transformation of determinant 1, thus $\det(\mathbf{G}) = \prod Q(\overline{\mathbf{b}}_i)$.

We have $\left\langle \mathbf{b}_i, \overline{\mathbf{b}}_i \right\rangle = \left\langle \overline{\mathbf{b}}_i, \overline{\mathbf{b}}_i \right\rangle$ because the Gram-Schmidt vectors are orthogonal, and

$$Q(\mathbf{b}_i) = \left\langle \mathbf{b}_i, \overline{\mathbf{b}}_i \right\rangle + \sum_{j=0}^{i-1} \mu_{i,j} \left\langle \mathbf{b}_i, \overline{\mathbf{b}}_j \right\rangle = Q\left(\overline{\mathbf{b}}_i\right) + \sum_{j=0}^{i-1} \mu_{i,j}^2 Q\left(\overline{\mathbf{b}}_j\right).$$

Clearly $Q(\mathbf{b}_i) \geq Q\left(\overline{\mathbf{b}}_i\right)$ because the quadratic form is positive definite, proving the second inequality of the claim.

Now, because the basis is (η, δ)-reduced

$$Q(\overline{\mathbf{b}}_i) \geq (\delta - \eta^2) Q(\overline{\mathbf{b}}_{i-1}) \geq (\delta - \eta^2)^{i-j} Q(\overline{\mathbf{b}}_j)\,,$$

thus, setting $\theta = (\delta - \eta^2)^{-1}$,

$$Q(\mathbf{b}_i) \leq Q(\overline{\mathbf{b}}_i) + \eta^2 \sum_{j=0}^{i-1} \frac{1}{(\delta - \eta^2)^{i-j}} Q(\overline{\mathbf{b}}_i) \leq \frac{\theta^{i+1} - 1}{\theta - 1} Q(\overline{\mathbf{b}}_i) \leq \frac{\theta^{i+1}}{\theta - 1} Q(\overline{\mathbf{b}}_i)\,.$$

Finally

$$\prod_{i=0}^{d-1} Q(\mathbf{b})_i \leq \frac{\theta^{d(d-1)/2}}{(\theta - 1)^d} \det(\mathbf{G}).$$

□

Lemma A.4. *Let $R \subset \mathbb{R}^d$ be the parallelogram containing all vectors $(x_0, \ldots, x_{d-1})$ which Algorithm* LatticeSampling *on page 26 may sample before rejection. Let $S \subset \mathbb{R}^d$ be the ellipsoid defined by $\mathbf{x}\mathbf{G}\mathbf{x}^t \leq B$. Let (η, δ) be the parameters used for lattice reduction and set $\theta = (\delta - \eta^2)^{-1}$, then*

$$\frac{\mathrm{vol}(S)}{\mathrm{vol}(R)} \geq \left(\frac{\pi}{4}\right)^{d/2} \frac{(\theta - 1)^{d/2}}{\theta^{d(d-1)/4}} \frac{1}{\Gamma(d/2 + 1)}.$$

Proof. By definition

$$\mathrm{vol}(R) \leq 2^d B^{d/2} \prod_{i=0}^{d-1} \sqrt{\mathbf{H}_{i,i}}.$$

The volume of the unit ellipsoid $\mathbf{x}\mathbf{G}\mathbf{x}^t \leq 1$ is

$$\frac{\pi^{d/2}}{\Gamma(d/2 + 1)} \frac{1}{\sqrt{\det(\mathbf{G})}} = \frac{\pi^{d/2} \sqrt{\det(\mathbf{H})}}{\Gamma(d/2 + 1)},$$

thus

$$\mathrm{vol}(S) = \frac{\pi^{d/2} B^{d/2} \sqrt{\det(\mathbf{H})}}{\Gamma(d/2 + 1)}.$$

But the Gram matrix $\mathbf{H}$ is (η, δ)-reduced, thus by Lemma A.3

$$\frac{\mathrm{vol}(S)}{\mathrm{vol}(R)} \geq \left(\frac{\pi}{4}\right)^{d/2} \frac{(\theta - 1)^{d/2}}{\theta^{d(d-1)/4}} \frac{1}{\Gamma(d/2 + 1)}.$$

□

In particular, for $d = 4$, $\eta = 0.51$ and $\delta = 0.75$ we obtain an estimate for the accepting probability of ≈ 0.039. Counter-intuitively, a larger value of δ leads to a worse bound, but this is due to the simplifications in the minoration: a tighter version of Lemma A.3 shows the probability is indeed increasing with δ. Our experiments show the actual probability is even better on average: close to $1/3$.

APPENDIX B

Tables of constants

General Scheme Parameters. The following table gives the constants used as general scheme parameters.

Constant	NIST Level			Formula
	I	III	V	
λ	128	192	256	
p	See Chapter 5			
f	248	376	500	$\mathsf{DyadicValuation}(p+1)$
D_{mix}	See formula			$\texttt{next\_prime}(2^{\lceil \log_2(4\lambda) \rceil})$
e_{rsp}	126	192	253	$\lceil \log_2(\sqrt{p}) \rceil$
D_{rsp}	2^{126}	2^{192}	2^{253}	$2^{e_{\mathsf{rsp}}}$
D_{chl}	2^{248}	2^{376}	2^{500}	2^f
e_{chl}	122	184	247	$f - e_{\mathsf{rsp}}$

Lattice reduction. The following table gives the constants used in lattice reduction.

Constant	NIST Level			Formula
	I	III	V	
δ	0.99			
η	0.51			
Floating-point min. precision	24 bits			
Floating-point min. exponent	20 bits			

Quaternion Computations and Ideal-to-Isogeny Translation. The following table gives the constants used in quaternion computations and IdealToIsogeny.

Constant	NIST Level			Formula
	I	III	V	
`QUAT_primality_num_iter`	32	32	32	$\lceil -\log_4(2^{-64}) \rceil$
`QUAT_equiv_bound_coeff`	64	64	64	$2^{1+\texttt{QUAT\_repres\_bound\_input}/4}$
`QUAT_repres_bound_input`	20	21	21	$\left\lceil \log_2(\log_{1-1/(64\log_2(p))}(2^{-64})) \right\rceil$
`FINDUV_box_size`	2	3	3	$2 + \lfloor (\log_2 p - f)/4 \rfloor$
n_{orders}	6	7	6	Experimental, see Section 9.3.2
(q_t)	$\begin{pmatrix} 5, 17, 37 \\ 41, 53, 97 \end{pmatrix}$	$\begin{pmatrix} 5, 13, 17, 41 \\ 73, 89, 97 \end{pmatrix}$	$\begin{pmatrix} 5, 37, 61 \\ 97, 113, 149 \end{pmatrix}$	See Section 3.2.1.1
`QUAT_prime_cofactor`	$2^{252} + 65$	$2^{384} + 369$	$2^{506} + 51$	Smallest prime larger that $2^{\lceil \log_2(p) \rceil}$

Elliptic Curve Computations. The following table gives the constants used in elliptic curve computations.

Constant	NIST Level			Formula
	I	III	V	
TORSION_2POWER_BYTES	32	48	63	$\lfloor (f+7)/8 \rfloor$

Encoding. The following table gives the constants used in encoding various objects.

Constant	NIST Level			Formula
	I	III	V	
HASH_ITERATIONS	64	256	512	$2^{32 \cdot \lceil \log_2 p/64 \rceil - (f - e_{\text{rsp}})}$
FP_ENCODED_BYTES	32	48	64	$8 \lfloor (\log_2(p) + 63)/64 \rfloor$
FP2_ENCODED_BYTES	64	96	128	$2 \cdot$ FP_ENCODED_BYTES
EC_CURVE_ENCODED_BYTES	64	96	128	$2 \cdot$ FP_ENCODED_BYTES
PUBLICKEY_BYTES	65	97	129	$1 +$ EC_CURVE_ENCODED_BYTES
SECRETKEY_BYTES	353	529	701	PUBLICKEY_BYTES $+ 4 \cdot$ FP_ENCODED_BYTES $+ 4 \cdot$ TORSION_2POWER_BYTES
SIGNATURE_BYTES	148	224	292	$4 +$ EC_CURVE_ENCODED_BYTES $+ 4 \lfloor (e_{\text{rsp}} + 7)/8 \rfloor + \lambda/8$