SQIsign

Algorithm specifications and supporting documentation

Version 3.0

- Marius A. Aardal, *Aarhus University, Denmark*
- Gora Adj, *Technology Innovation Institute, UAE*
- Diego F. Aranha, *Aarhus University, Denmark*
- Andrea Basso, *IBM Research Europe, Switzerland*
- Giacomo Borin, *IBM Research Europe and University of Zurich, 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*
- Pierrick Dartois, *DGA-MI, Bruz, France*
- Luca De Feo, *IBM Research Europe, Switzerland*
- Max Duparc, *EPFL, Switzerland*
- Thomas Espitau, *PQShield SAS, France*
- Jonathan Komada Eriksen, *KU Leuven, Belgium*, previously *NTNU, Norway*
- Tako Boris Fouotsa, *The University of Manchester, UK*
- Décio Luiz Gazzoni Filho, *Technology Innovation Institute, UAE* and *State University of Londrina, Brazil*
- Basil Hess, *IBM Research Europe, Switzerland*
- Riccardo Invernizzi, *KU Leuven, Belgium*
- 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 Birmingham, UK*
- Michael Meyer, *University of Regensburg, Germany*
- Marzio Mula, *University of the Bundeswehr Munich, 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*
- Frederik Vercauteren, *KU Leuven, Belgium*
- Alexandre Wallet, *PQShield Ltd., UK*
- Benjamin Wesolowski, *CNRS* and *ENS de Lyon, France*
- Wessel van Woerden, *PQShield B.V., Netherlands*

September 1, 2026

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 with an overview of the mathematical objects used in SQIsign, a design rationale and a high-level description of the scheme. Then Chapter 3 gives a detailed and unambiguous description of SQIsign itself (key generation, signing, and verification), though, in the interest of clarity, not every subroutine is described in this chapter. The following Chapter 4 picks up the task of unambiguously defining every single routine invoked in SQIsign, and describing implementations for every non-obvious one.

The following chapters discuss the choice of parameters (Chapter 5), binary encodings (Chapter 6), performance (Chapter 7), security (Chapter 8), and heuristics and failure cases (Chapter 9).

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 $5\times$ smaller than Falcon (to be standardized as FN-DSA), and $12\times$ 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 optimized 64-bit Intel implementation at NIST security level I has a runtime of about 27.6 ms for signing and 3.6 ms for verification on an Intel Core i7-13700K (see Table 2 and Chapter 7). Continued improvements from earlier versions of SQIsign place it in the same ballpark as many other post-quantum signature candidates.
- ⊖ **A complex signing procedure.** Though the present iteration of the specification has made strides in simplifying and streamlining it, the main drawback of SQIsign remains 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 2).

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]

- ⊕ **Tight provable security.** SQIsign is proven EUF-CMA secure assuming the hardness of a version *with hints* of the endomorphism ring problem. As we survey in Section 8.1, the reduction is tight if one is willing to accept standard heuristics.

[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, the underlying algorithms for the finite-field, elliptic-curve, pairing, isogeny arithmetic and lattice reductions have been implemented in constant-time. Also, all the integers involved in the signing operation have effective bounds.

[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	83	270	200
NIST-III	129	417	306
NIST-V	169	549	406

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	32.2	93.9	12.1
NIST-III	102.1	320.3	31.5
NIST-V	214.5	674.7	77.6

⊕ **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.

⊖ **Concrete security in flux.** Though the previous iteration of this specification claimed that "The complexity of the best algorithms for [solving the endomorphism ring] problem has been very stable", one cannot but recognize that things have changed. While this document was being drafted, one of the authors came to the realization [Wes26] that a well-known family of attacks could be improved by an exponential factor, lowering the overall complexity of computing endomorphism rings from $p^{1/2+o(1)}$ to $p^{1/3+o(1)}$.

While the new attack does not compromise the viability of SQISIGN, it casts it in a state of uncertainty about the exact security of its parameters. The time incurred between the discovery of the attack and the completion of this document has objectively been too short to form a clear picture of the ramifications of the new attack. At the time of writing, we are aware of a handful of crucial optimizations to [Wes26] that make it a more credible threat than initially thought. We briefly describe these optimizations in Section 8.2.

At the same time, Wesolowski's attack suffers from requiring an exponential amount of memory. The only credible way to implement it at the scale needed for breaking cryptographic parameters is by a time-memory compromise of $t^2 w = \tilde{O}(p)$, where t denotes time and w memory. This is to be compared with previous approaches which had $t^2 = \tilde{O}(p)$.

It is this team's opinion that, in light of the new attack, SQISIGN's parameters need to be revised; however, ignoring the impact of memory and scaling them by a full factor of $3/2$ would be an overreaction: SQISIGN's running time increases more than linearly with the size of parameters, hence overestimating the power of the new attack would result in an unnecessary degradation in performance, and thus increased costs when it is deployed at industrial scale.

The parameters put forth in the present specification represent a middle-of-the-road approach, motivated by the time-memory tradeoff discussed in Section 8.2. We are optimistic that, barring new improvements such as a memory-free-$p^{1/3+o(1)}$ attack, they will be found to meet and even surpass the security targets mandated by NIST.

⊕ **Abundant and fine-grained parameters.** And if our predictions had to fail, there is at least relief in knowing that replacement parameters are abundant and permit a fine-grained tuning of SQISIGN. The only requirement to instantiate (this specification's version of) SQISIGN is having a prime number of the form $p = c \cdot 2^f - 1$, with c as small as possible (performance of signing sensibly degrades as c grows). Only in the interval $256 < \log_2(p) < 384$, seventeen choices of prime with $c < 20$ exist. We give in Appendix P a list of alternative parameters that constitute interesting targets for optimization, should the present parameters turn out not to offer the best performance/security compromise.

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 $\varphi_{\mathsf{com}}^{\mathrm{cheat}} : E_{\mathsf{pk}} \to E_{\mathsf{com}}$. 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 $\varphi_{\mathsf{rsp}}^{\mathrm{cheat}} = \varphi_{\mathsf{chl}} \circ \widehat{\varphi_{\mathsf{com}}^{\mathrm{cheat}}} : E_{\mathsf{com}} \to E_{\mathsf{chl}}$ (where $\widehat{\varphi_{\mathsf{com}}^{\mathrm{cheat}}} : E_{\mathsf{com}} \to E_{\mathsf{pk}}$ is the *dual* of $\varphi_{\mathsf{com}}^{\mathrm{cheat}}$).

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 the next chapters 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 $\varphi_{\mathsf{com}} : E_0 \to E_{\mathsf{com}}$ 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 $E_0 \to E$. We summarize this point of view in Section 1.2.

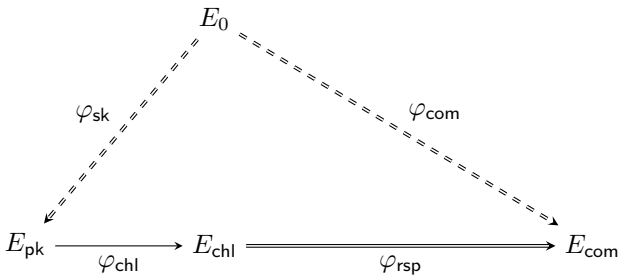


FIGURE 1. High-level conceptual overview of the isogenies constructed in SQISIGN's underlying Σ-protocol. Dashed arrows are the signer's secrets; solid arrows are public. The challenge φ_{chl} is sampled (in a public-coin manner) by the verifier; all other arrows are constructed by the signer. Double arrows indicate isogenies evaluated using a two-dimensional ideal-to-isogeny algorithm.

1.3. Differences between the round-2 and round-1 SQISIGN submissions

Text copied verbatim from the round-2 specification of SQISIGN

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" $\varphi_{\mathsf{rsp}} : E_{\mathsf{com}} \to E_{\mathsf{chl}}$ 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. Differences from the round-2 SQISIGN submission

The changes from the round-2 submission have been relatively minor by SQISIGN standards. The most important ones are:

(1) **Increased parameter sizes** in reaction to the recently discovered attack [Wes26]. The increase in size leads to a 1.5-$2\times$ slow-down in verification, but is more than offset in key-generation and signing by the improvements listed below.

(2) **Entirely redesigned quaternions.** The previous iteration relied on standard multiple-precision integer linear algebra algorithms for computations in quaternion algebras. Bounding the size of integers in the computations was notoriously difficult and early attempts at fixed-precision integer arithmetic introduced unbearable slow-downs [KLKL26]. The present iteration introduces a new representation for quaternionic ideals called *inert representation* [Ler26; LS26]. Carefully redesigning algorithms, we are now able to set tight bounds on the growth of integers, replacing multiple-precision with fixed-precision without incurring performance losses.

(3) **Dropped floating point arithmetic and improved lattice reduction.** New lattice reduction algorithms tailored to the quaternion algebra $B_{p,\infty}$ drop the requirement for floating point types and improve performance. Their outputs enforce canonical forms for quaternion lattices, simplifying reproducibility of test vectors and enabling the possibility of deterministic signatures.

(4) **Adopted a new ideal-to-isogeny algorithm.** As introduced in [BCE+25] and improved in [DLS26], the new algorithm offers considerable speed-ups in key-generation and signing, reduces complexity and memory footprint, and brings failure cases down to a statistically negligible number.

(5) **Improved isogeny formulas** introduced in [Dar25; Dup25] simplify the code and give moderate speed-ups.

(6) **Changed the distribution of responses in the Σ-protocol.** This is the only cryptographically-relevant change. The round-2 version of SQISIGN defined the challenge isogeny φ_{chl} to be a 2-isogeny walk of length $\approx 2\lambda$, then sampled the response isogeny φ_{rsp} as a random isogeny of degree $\approx \sqrt{p}$. This allowed for the possibility that φ_{chl} and φ_{rsp} share a common suffix, significantly complicating the computation and representation of φ_{rsp}. Additionally, soundness of the Σ-protocol could only be ensured by having $2\lambda > \log(p)/2$.

In the present revision, we have modified φ_{chl} to have length λ and φ_{rsp} to have **odd** degree $\approx \sqrt{p2^{\lambda/2}}$, ensuring that φ_{chl} and φ_{rsp} have no common factor. Soundness follows immediately and the computation of φ_{rsp} is more streamlined, leading to a noticeable reduction in code size and simpler algorithms. Overall, this change leads to an increase in signature size by $\lambda/8$ bytes and a minor speed-up in verification.

This change also affects the provable security of SQISIGN. In both cases, unforgeability reduces to the endomorphism ring problem with *hints* whose degrees are distributed like $\deg \varphi_{\mathsf{rsp}}$. Having changed φ_{rsp}, the hints change accordingly, thus security assumptions for the two versions are technically incomparable. In practice, the difference between arbitrary and odd degree is minimal, and, if anything, increasing the degree of the hints could make the problem harder. There is no reason to believe that the change makes SQISIGN less secure. We discuss this question in depth in Sections 8.1 and 8.2.

Additionally, the following improvements have been made to the reference implementation:

(1) **Dropped the dependency on external libraries.** Formerly SQISIGN relied on GMP for multi-precision integers and DPE for double-precision-exponent floating points. The former has been replaced by a custom-made fixed-precision integer library; the latter is no longer needed, as floating-point numbers have been completely removed.

(2) **Added support for various platforms.** The code now compiles and runs on x86_64, arm64 and Cortex-M4 platforms, with assembly optimizations for broadwell and AARCH64 architectures.

Overall, the present revision significantly accelerates SQISIGN, at the same time making it simpler to analyse and easier to implement. Though we could not achieve a fully constant-time implementation at this time, the changes to integer arithmetic were a necessary first step and pave the way for it.

CHAPTER 2

Protocol Overview

Like ECDSA, ML-DSA, and many others, SQIsign belongs to the family of Fiat-Shamir signatures [FS87]: it consists of a Σ-protocol made non-interactive by replacing the challenge with the output of a random oracle.

Underlying SQIsign is the theory of *supersingular elliptic curves*[1] and their *morphisms*. SQIsign's public key is a randomly chosen supersingular elliptic curve and, informally, SQIsign's Σ-protocol is a proof of knowledge of the *endomorphisms* of the public key. Finding even a single *non-scalar endomorphism* of a random supersingular curve is believed to be exponentially hard, even for quantum computers, and is considered to be the most fundamental problem in isogeny-based cryptography [EHL+18; PW24; Wes22].

SQIsign leverages *Deuring's correspondence*, a mathematical equivalence between supersingular elliptic curves and *ideal classes of a quaternion algebra*. Quaternionic ideals are the secret data in SQIsign: they encode the secret key, as well as the secret randomness of the Σ-protocol. Elliptic curves and morphisms are the public data: they encode public keys, commitments, and responses of the Σ-protocol. The Deuring correspondence defines bijections **from ideals to morphisms** and **from ideal classes to supersingular elliptic curves**. The algorithms presented in this specification make the correspondence explicit.

2.1. Mathematical overview

To help the reader navigate this document, we now list the key facts about supersingular elliptic curves and quaternion algebras. These are not introductory lecture notes, nor a survey paper. The reader interested in learning more about the mathematical foundations of SQIsign is invited to read [Mil86; Sil09; Voi21].

Notation. 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 write $[a\,..\,b[$ for the set of integers between a (included) and b (excluded).

We denote matrices with bold capital letters, e.g., $\mathbf{M}$, and vectors with bold small letters, e.g., $\mathbf{v}$. 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 write $[n]P$ for the n-th scalar multiple of an elliptic point P. If $\mathbf{M} = \left(\begin{smallmatrix} a & b \\ c & d \end{smallmatrix}\right)$ is a 2×2 integer matrix and $\mathcal{B} = (P, Q)$ a pair of points on an elliptic curve, we write

$$\mathcal{B} \cdot \mathbf{M} = (P, Q) \cdot \mathbf{M} = \big([a]P + [c]Q, [b]P + [d]Q\big). \tag{1}$$

Isogenies between elliptic curves are denoted with Greek letters, e.g., φ, ψ, τ; endomorphisms often use θ. If θ is an endomorphism of E and (P, Q) a basis of the N-torsion $E[N]$, we write $\mathbf{M}_\theta$ for the matrix in $\mathbb{M}_2(\mathbb{Z}/N\mathbb{Z})$ such that

$$\big(\theta(P), \theta(Q)\big) = (P, Q) \cdot \mathbf{M}_\theta. \tag{2}$$

By this convention, $\mathbf{M}_{\theta\circ\eta} = \mathbf{M}_\theta \mathbf{M}_\eta$.

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. More specific notation used in this document is defined in Tables 3 and 4.

Supersingular elliptic curves.

Fact 1) An elliptic curve defined over a finite field k of characteristic p is supersingular if and only if $\#E(k)$ is congruent to 1 modulo p.

[1] Supersingular elliptic curves are the only algebraic curves featured in this document. Since there is no ambiguity, we will refer to them interchangeably as "supersingular curves", "elliptic curves", or just "curves".

Fact 2) A morphism φ from an elliptic curve E to an elliptic curve E', written $\varphi\colon E \to E'$, is an algebraic map from E to E' sending the point at infinity of E to the point at infinity of E'. All morphisms are group morphisms, i.e., $\varphi(P+Q) = \varphi(P) + \varphi(Q)$ for any points $P, Q \in E$.

Fact 3) There exists a unique constant morphism from E to E', which maps any point of E to the identity element of E'. Non-constant morphisms are called *isogenies*.

Fact 4) Morphisms between the same pair of curves can be added: $\varphi + \psi$ is the unique morphism such that $(\varphi+\psi)(P) = \varphi(P) + \psi(P)$. The group of all morphisms from E to E' is written $\mathrm{Hom}(E, E')$.

Fact 5) A morphism from a curve to itself is called an *endomorphism*. For any integer $n \in \mathbb{Z}$ there exists an endomorphism $[n]\colon E \to E$ that maps every point of E to its n-th multiple. They are called *scalar multiplications* or *scalar endomorphisms*. Any other endomorphism is called *non-scalar*.

Fact 6) Isogenies admitting a (necessarily two-sided) inverse isogeny are called *isomorphisms*. Endomorphisms that are also isomorphisms are called *automorphisms*.

Fact 7) Endomorphisms can be composed: if φ and ψ are endomorphisms of E, then $\varphi \circ \psi$ and $\psi \circ \varphi$ are too. Composition distributes over addition and $[1]$ acts as multiplicative identity, thus the set of all endomorphisms forms a ring, the *endomorphism ring*, which is denoted by $\mathrm{End}(E)$.

Fact 8) The *degree* of an isogeny φ, denoted by $\deg\varphi$, is a strictly positive integer. By convention $\deg[0] = 0$. The degree is multiplicative with respect to composition: $\deg(\varphi\circ\psi) = \deg\varphi \cdot \deg\psi$. In particular, isomorphisms are precisely the isogenies of degree 1. The degree is a positive definite quadratic form on $\mathrm{Hom}(E, E')$; in particular $\deg([n]\varphi) = n^2 \deg\varphi$.

Fact 9) An isogeny φ between supersingular elliptic curves over a field of characteristic p is *separable* if and only if $p \nmid \deg(\varphi)$. If φ is separable then $\#\ker\varphi = \deg\varphi$. An isogeny φ is *inseparable* (i.e., not separable), if and only if it factors through the *Frobenius isogeny* $(x, y) \mapsto (x^p, y^p)$.

Fact 10) Every morphism $\varphi\colon E \to E'$ has a unique *dual morphism* $\widehat{\varphi}\colon E' \to E$ such that $\deg\widehat{\varphi} = \deg\varphi$ and $\varphi\circ\widehat{\varphi} = [\deg\varphi]_{E'}$ and $\widehat{\varphi}\circ\varphi = [\deg\varphi]_E$. Duality commutes with addition: $\widehat{\varphi+\psi} = \widehat{\varphi} + \widehat{\psi}$, and is contravariant with respect to composition: $\widehat{\varphi\circ\psi} = \widehat{\psi}\circ\widehat{\varphi}$.

Fact 11) Two elliptic curves are called *isogenous* if there is an isogeny between them. Two curves over the same finite field are isogenous (over that field) if and only if they have the same number of points.

Quaternion algebras.

Fact 12) A *quaternion algebra* B over $\mathbb{Q}$ is a four-dimensional $\mathbb{Q}$-algebra with a $\mathbb{Q}$-basis $1, \mathbf{i}, \mathbf{j}, \mathbf{k}$ such that

$$\mathbf{i}^2 = a, \quad \mathbf{j}^2 = b, \quad \mathbf{ij} = \mathbf{k} = -\mathbf{ji},$$

for some integers a, b.

Fact 13) The *conjugate* of a quaternion $\alpha = x + y\mathbf{i} + z\mathbf{j} + t\mathbf{k}$ is the quaternion $\bar{\alpha} = x - y\mathbf{i} - z\mathbf{j} - t\mathbf{k}$. Its *reduced norm* is $\mathrm{nrd}(\alpha) = \mathrm{nrd}(\bar{\alpha}) = \alpha\bar{\alpha}$ and its *reduced trace* is $\mathrm{tr}(\alpha) = \mathrm{tr}(\bar{\alpha}) = \alpha + \bar{\alpha}$. In this document we omit "reduced" and simply call them *norm* and *trace*. The reduced norm is a positive definite quadratic form on B.

Fact 14) A *lattice* in a quaternion algebra B is a $\mathbb{Z}$-module of rank 4, i.e., one containing a vector-space basis for B. For example, the $\mathbb{Z}$-module generated by $\left(1/3, \mathbf{i}, \mathbf{j}, \frac{\mathbf{i}+\mathbf{k}+1}{2}\right)$ is a lattice.

Fact 15) A lattice that is also a subring of B is called an *order* in B. An order in B is *maximal* if it is not strictly contained in any other order of B. For example, the set $\{1, \mathbf{i}, \mathbf{j}, \mathbf{k}\}$ generates an order, though not a maximal one.

Fact 16) If $\mathcal{O}$ is an order in a quaternion algebra, an *integral left $\mathcal{O}$-ideal* is a lattice contained in $\mathcal{O}$ and stable by left-multiplication by elements of $\mathcal{O}$. In this document we call these *$\mathcal{O}$-ideals*, or simply *ideals* when the order $\mathcal{O}$ is clear from context. For example $\mathcal{O}\alpha$ is an $\mathcal{O}$-ideal for any $\alpha \in \mathcal{O}\backslash\{0\}$; an ideal of this form is called *principal*.

Fact 17) The *right-order* of an ideal I is the unique order

$$\mathcal{O}_R(I) = \{\alpha \in B \mid I\alpha \subset I\}.$$

In this document we use the non-standard notation $I\colon \mathcal{O} \to \mathcal{O}'$ to indicate that I is a left $\mathcal{O}$-ideal with right-order $\mathcal{O}'$.

Fact 18) The *conjugate* of an ideal $I\colon \mathcal{O} \to \mathcal{O}'$ is the ideal $\bar{I}\colon \mathcal{O}' \to \mathcal{O}$ defined by

$$\bar{I} = \{\bar{\alpha} \mid \alpha \in I\}.$$

Fact 19) If $I\colon \mathcal{O} \to \mathcal{O}'$ and $J\colon \mathcal{O}' \to \mathcal{O}''$ are ideals, their *product* $IJ\colon \mathcal{O} \to \mathcal{O}''$ is the ideal

$$IJ = \left\{ \sum_{i=0}^{n} \alpha_i \beta_i \mid \alpha_i \in I, \beta_i \in J \right\}$$

generated by all finite products $\alpha\beta$ of elements $\alpha \in I$, $\beta \in J$. Ideal multiplication is associative.

Fact 20) The *norm of an ideal* is the greatest common divisor of the norms of its elements:

$$\mathrm{nrd}(I) = \gcd(\mathrm{nrd}(\alpha) \mid \alpha \in I).$$

The norm is multiplicative with respect to products: $\mathrm{nrd}(IJ) = \mathrm{nrd}(I)\,\mathrm{nrd}(J)$.

Fact 21) If $I\colon \mathcal{O} \to \mathcal{O}'$ and $\alpha \in B^\times$, then $J = I\alpha$ is an $\mathcal{O}$-ideal with right-order $\alpha^{-1}\mathcal{O}'\alpha$, which is isomorphic to $\mathcal{O}'$. In this situation I and J are said to be *equivalent ideals*.

Fact 22) An $\mathcal{O}$-*ideal class* is an equivalence class of $\mathcal{O}$-ideals under the relation $J = I\alpha$ where $\alpha \in B^\times$. The ideal class of I is denoted by $\mathrm{cls}(I)$. In particular, all ideals in $\mathrm{cls}(I)$ have isomorphic right-orders. Ideal classes are compatible with multiplication: If $I\colon \mathcal{O} \to \mathcal{O}'$ and $J\colon \mathcal{O}' \to \mathcal{O}''$, the ideal class $\mathrm{cls}(IJ)$ only depends on $\mathrm{cls}(I)$ and $\mathrm{cls}(J)$, which justifies writing $\mathrm{cls}(IJ) = \mathrm{cls}(I)\,\mathrm{cls}(J)$.

The Deuring correspondence.

Fact 23) When E is a supersingular elliptic curve defined over a finite field $\mathbb{F}_{p^2}$, then $\mathrm{End}(E) \otimes \mathbb{Q}$ is a quaternion algebra, and the endomorphism ring $\mathrm{End}(E)$ is an order in B. From now on, we fix a curve E_0 and an order $\mathcal{O}_0$ in a suitable quaternion algebra B with a choice of isomorphism $\iota_0\colon \mathrm{End}(E_0) \to \mathcal{O}_0$.

Fact 24) $\mathcal{O}_0$-ideal classes are in bijection with isomorphism classes of supersingular curves. The bijection is given explicitly in Definition 2.2.3. If $\mathrm{cls}(I)$ is an $\mathcal{O}_0$-ideal class, we write $\mathrm{cls}(I) \star E_0$ for the isomorphism class associated to I by this bijection. The endomorphism ring $\mathrm{End}(\mathrm{cls}(I) \star E_0)$ is isomorphic to the right-order of any representative of $\mathrm{cls}(I)$.

Fact 25) If $\mathrm{cls}(I)$ is an $\mathcal{O}_0$-ideal class and $E \in \mathrm{cls}(I) \star E_0$ is a representative of the associated isomorphism class of supersingular elliptic curves, the ideals in $\mathrm{cls}(I)$ are in bijection with isogenies $\varphi\colon E_0 \to E$ up to post-composition with automorphisms of E, i.e., φ and $\alpha \circ \varphi$ are considered equivalent isogenies when α is any automorphism of E. The bijection is given explicitly in Definition 2.2.3. Unless $j(E) \in \{0, 1728\}$, the only automorphisms of E are $[\pm 1]$, thus the equivalence only collapses φ and $-\varphi$.

Fact 26) We write $I \star E_0$, or more compactly φ_I, for the isogeny $\varphi_I\colon E_0 \to E$ (up to post-automorphisms) associated to I, for any choice of $E \in I \star E_0$. Then $\mathrm{nrd}(I) = \deg \varphi_I$.

Fact 27) Let $\varphi_I\colon E_0 \to E_I$ and $\varphi_J\colon E_0 \to E_J$ be isogenies associated to $\mathcal{O}_0$-ideals I and J respectively. The lattice $\Lambda = \bar{I}J$ is in bijection with $\mathrm{Hom}(E_I, E_J)$. The bijection is given explicitly in Proposition 2.2.7.

2.2. Mathematical definitions

The following constants and definitions are necessary to correctly implement SQIsign.

2.2.1. Constants

A SQIsign parameter set is defined by a single prime of the form $p = c \cdot 2^f - 1$, for some exponent $f > 1$ and a small odd cofactor c. See Chapter 5 for the list of specified primes.

Given p, we define a few derived mathematical constants that occur repeatedly throughout the document:

- The finite field $\mathbb{F}_{p^2} = \mathbb{F}_p[I]/(I^2 + 1) = \mathbb{F}_p[\mathrm{i}]$;
- The elliptic curve $E_0\colon y^2 = x^3 + x$ defined over $\mathbb{F}_{p^2}$;
- The quaternion algebra $B_{p,\infty}$ over $\mathbb{Q}$ defined by $\mathbf{i}^2 = -1$, $\mathbf{j}^2 = -p$ and $\mathbf{k} = \mathbf{ij} = -\mathbf{ji}$;
- The maximal order $\mathcal{O}_0 \subset B_{p,\infty}$ generated by $\left(1, \mathbf{i}, \frac{\mathbf{i}+\mathbf{j}}{2}, \frac{1+\mathbf{ij}}{2}\right)$.

All parameters are summarised in Table 3.

TABLE 3. Mathematical constants

Names	Meaning	Notes
p	A prime of the form $p = c \cdot 2^f - 1$ with $f > 1$ and c odd	
$\mathbb{F}_p$	Finite field with p elements	
$\mathbb{F}_{p^2}$	Finite field with p^2 elements defined as $\mathbb{F}_{p^2} = \mathbb{F}_p[I]/(I^2+1) = \mathbb{F}_p[i]$	
i	The standard generator of $\mathbb{F}_{p^2} \,/\, \mathbb{F}_p$	$\mathsf{i}^2 = -1$
E_0	The supersingular elliptic curve $y^2 = x^3 + x$ over $\mathbb{F}_{p^2}$	
$B_{p,\infty}$	The quaternion algebra over $\mathbb{Q}$ with $\mathbf{i}^2 = -1$, $\mathbf{j}^2 = -p$, $\mathbf{k} = \mathbf{ij} = -\mathbf{ji}$	
$1, \mathbf{i}, \mathbf{j}, \mathbf{k}$	The standard basis of $B_{p,\infty}$ over $\mathbb{Q}$	
$\mathcal{O}_0$	The maximal order of $B_{p,\infty}$ generated by $\left(1, \mathbf{i}, \frac{\mathbf{i}+\mathbf{j}}{2}, \frac{1+\mathbf{ij}}{2}\right)$	$\mathrm{End}(E_0) \simeq \mathcal{O}_0$

2.2.2. The Deuring correspondence

The choice of p above ensures that $\mathrm{End}(E_0) \simeq \mathcal{O}_0$. We explicitly define this isomorphism as follows.

Definition 2.2.1. E_0 is equipped with the endomorphisms:

- $\iota\colon (x, y) \mapsto (-x, \mathsf{i}y)$,
- $\pi\colon (x, y) \mapsto (x^p, y^p)$.

The isomorphism $\iota_0\colon \mathrm{End}(E_0) \to \mathcal{O}_0$ is defined by linearly extending the mapping $\iota_0\colon \iota \mapsto \mathbf{i}$ and $\iota_0\colon \pi \mapsto \mathbf{j}$.

We now define the correspondence between $\mathcal{O}_0$-ideals and isogenies of E_0.

Definition 2.2.2 (Quotient isogeny)**.** Let E be an elliptic curve and let $G \subset E$ be a finite subgroup. Let $\varphi\colon E \to E'$ be a separable isogeny such that $\ker\varphi = G$. The isogeny φ is unique up to isomorphisms of E' and it is called a *quotient isogeny of E by G*. The isomorphism class of E is denoted by E/G and is called a *quotient (curve) of E by G*.

The definitions below are restricted to ideals of norm not divisible by p and separable isogenies; they can be extended to any ideal and isogeny, however this is not necessary for the purpose of this specification.

Definition 2.2.3 (Deuring correspondence)**.** Let I be an $\mathcal{O}_0$-ideal and assume $p \nmid \mathrm{nrd}(I)$. Define the subgroup $E_0[I] \subset E_0(\overline{\mathbb{F}}_p)$

$$E_0[I] = \left\{ P \in E_0(\overline{\mathbb{F}}_p) \;\middle|\; \omega(P) = 0 \text{ for all } \omega \in \iota_0^{-1}(I) \right\}. \tag{3}$$

The Deuring correspondence is the equivalence of categories mapping:

- The $\mathcal{O}_0$-ideal class $\mathrm{cls}(I)$ to the isomorphism class $\mathrm{cls}(I) \star E_0 = E_0/E_0[I]$,
- The $\mathcal{O}_0$-ideal I to $I \star E_0$, the (post-isomorphism class of) quotient isogenies $\varphi_I\colon E_0 \to E_0/E_0[I]$.

Proposition 2.2.4. *If $\varphi\colon E_0 \to E$ is an isogeny, the inverse of the Deuring correspondence maps φ to the ideal*

$$I = \iota_0(\mathrm{Hom}(E, E_0)\varphi) = \{\iota_0(\psi \circ \varphi) \mid \psi \in \mathrm{Hom}(E, E_0)\}. \tag{4}$$

Indeed, for any choice of representative of $\mathrm{cls}(I) \star E_0$, there exists an isomorphism $\alpha\colon E \to \mathrm{cls}(I) \star E_0$ such that $\varphi_I = I \star E_0 = \alpha \circ \varphi$.

Definition 2.2.5 (Pushout)**.** Letting $\varphi\colon E \to E'$ and $\psi\colon E \to E''$ be separable isogenies, the *pushout* of (φ, ψ) is the quotient isogeny $\chi\colon E \to E/G$ where $G = \ker\varphi \cup \ker\psi$. The pushout is unique up to post-composition by isomorphisms.

Proposition 2.2.6. *Let I, J be $\mathcal{O}_0$-ideals and φ_I, φ_J associated isogenies. Then $\varphi_{I\cap J}$ is a pushout of (φ_I, φ_J).*

Proposition 2.2.7. *Let $\varphi_I\colon E_0 \to E_I$ and $\varphi_J\colon E_0 \to E_J$ be isogenies and I, J be the corresponding ideals. The lattice $\Lambda = \bar{I}J$ is in bijection with $\mathrm{Hom}(E_I, E_J)$ by linearly extending the mapping*

$$\begin{aligned} \{\bar{\alpha}\beta \mid \alpha = \iota_0(\psi \circ \varphi_I) \in I, \beta = \iota_0(\chi \circ \varphi_J) \in J\} &\to \mathrm{Hom}(E_I, E_J), \\ \bar{\alpha}\beta = \iota_0\left(\widehat{\varphi}_I \circ \widehat{\psi} \circ \chi \circ \varphi_J\right) &\mapsto \widehat{\chi} \circ \psi. \end{aligned} \tag{5}$$

We write $\Delta : \bar{I}J \to \mathrm{Hom}(E_I, E_J)$ for this bijection.

We summarise in Table 4 the notations pertaining to the Deuring correspondence.

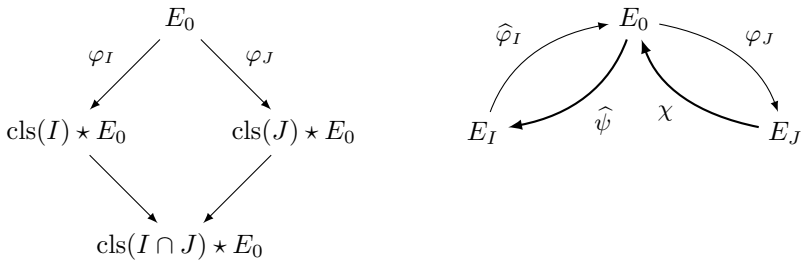


FIGURE 2. Illustration of a pushout (left) and of the bijection $\bar{I}J \simeq \mathrm{Hom}(E_I, E_J)$ of Proposition 2.2.7 (right).

TABLE 4. Notations for elliptic curves and the Deuring correspondence.

Notation	Explanation
E	An elliptic curve
E_A	An elliptic curve in Montgomery form $y^2 = x^3 + Ax^2 + x$
$j(E)$	The j-invariant of the elliptic curve E
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 or A
$[n]P$	The n-th scalar multiple of a point P
x_P and y_P	The x- and y-coordinates of a point $P \in E$
$\varphi\colon E \to E'$	Morphism φ with domain E and codomain E'
$I\colon \mathcal{O} \to \mathcal{O}'$	$\mathcal{O}$-ideal with right-order $\mathcal{O}'$
$\varphi \circ \psi\colon E \to E''$	Composition of morphism $\psi\colon E \to E'$ with $\varphi\colon E' \to E''$
$IJ\colon \mathcal{O} \to \mathcal{O}''$	Product of ideals $I\colon \mathcal{O} \to \mathcal{O}'$ and $J\colon \mathcal{O}' \to \mathcal{O}''$
$\deg \varphi$	Degree of morphism φ
$\mathrm{nrd}(\alpha), \mathrm{nrd}(I)$	Reduced norm of quaternion α and of ideal I
$\widehat{\varphi}$	Dual morphism to φ
$\bar{\alpha}, \bar{I}$	Conjugate of quaternion α and of ideal I
$\mathrm{Hom}(E, E')$	Additive group of morphisms $E \to E'$
$\mathrm{End}(E)$	Ring of endomorphisms of curve E
$\mathrm{cls}(I)$	Equivalence class of ideal I
$\mathrm{cls}(I) \star E_0$	Elliptic curve associated to $\mathrm{cls}(I)$ by the Deuring correspondence, up to isomorphism
$\varphi_I\colon E_0 \to E$	Isogeny associated to the $\mathcal{O}_0$-ideal I, up to post-composition with automorphisms of E
$I \star E_0$	Same as φ_I

2.3. The SQISIGN signature protocol

With the definitions and notations above, we now describe the SQISIGN signature scheme. Its parameters are a prime p and a *security parameter* λ, which together determine the security strength of the signature. The next chapter will define constraints on p and λ.

It is instructive to compare SQISIGN to Schnorr's signature scheme [Sch90; Sch91] —the underlying design of EdDSA [JL17]— as the high-level descriptions of the two schemes can be matched almost line-by-line. In Algorithm 2.1 we show the two algorithms side-by-side. For Schnorr's signature, we let G be a fixed generator of an elliptic curve group of prime order q; we write $a \cdot G$ for the elliptic curve scalar multiplication of G by an integer a.

2.3.1. Key generation

A secret key in SQISIGN is a random $\mathcal{O}_0$-ideal class $\mathrm{cls}(I_{\mathsf{sk}})$. Concretely, $\mathrm{cls}(I_{\mathsf{sk}})$ is represented by an $\mathcal{O}_0$-ideal I_{sk}, which for technical reasons must have odd prime norm. The associated public key is the supersingular elliptic curve $E_{\mathsf{pk}} = \mathrm{cls}(I_{\mathsf{sk}}) \star E_0$. Compare this to Schnorr's signature, where the secret key is an integer $\mathsf{sk} \in [0, q-1]$ and the public key is the group element $A_{\mathsf{pk}} = \mathsf{sk} \cdot G$.

Additionally, a public key in SQISIGN consists of a pair of points $P_{\mathsf{pk}}, Q_{\mathsf{pk}} \in E_{\mathsf{pk}}$ of order 2^λ forming a basis of $E_{\mathsf{pk}}[2^\lambda]$. This additional data has no analogue in Schnorr's signature. Its role is to correctly define the signing procedure.

The next chapter describes a procedure to select $P_{\mathsf{pk}}, Q_{\mathsf{pk}}$ so that they can be encoded in a single byte. Though the exact way $P_{\mathsf{pk}}, Q_{\mathsf{pk}}$ are chosen has no cryptographic relevance, every compliant implementation of SQISIGN must generate $P_{\mathsf{pk}}, Q_{\mathsf{pk}}$ as described in the next chapter.

2.3.2. Signature

Signing consists of the three steps sketched in the introduction: *commitment*, *challenge* and *response*.

Commitment is similar to key generation: it samples a uniformly random $\mathcal{O}_0$-ideal class $\mathrm{cls}(I_{\mathsf{com}})$, represented by an ideal I_{com} of odd (not necessarily prime) norm, then it computes the *commitment curve* $E_{\mathsf{com}} = \mathrm{cls}(I_{\mathsf{com}}) \star E_0$.

The challenge chl is computed by feeding the public key, the commitment curve and the message to a hash function $H : \{0,1\}^* \to [0\,..\,2^\lambda[$. The integer chl is then used to compute the point $K_{\mathsf{chl}} = P_{\mathsf{pk}} + [\mathsf{chl}]Q_{\mathsf{pk}}$, generating the kernel of the isogeny $\varphi_{\mathsf{chl}} : E_{\mathsf{pk}} \to E_{\mathsf{chl}} = E_{\mathsf{pk}}/\langle K_{\mathsf{chl}} \rangle$. The signer does not need to compute φ_{rsp}, though: instead, it computes the $\mathcal{O}_0$-ideal I_{chl} such that $E_0[I_{\mathsf{chl}}] = \langle \widehat{\varphi_{\mathsf{sk}}}(K_{\mathsf{chl}}) \rangle$.

Finally, the goal of the response is to prepare an isogeny $\varphi_{\mathsf{rsp}} : E_{\mathsf{chl}} \to E_{\mathsf{com}}$ statistically uncorrelated to φ_{sk}. To this effect:

- The signer prepares the response ideal $\Lambda = \overline{I_{\mathsf{sk}} \cap I_{\mathsf{chl}}} \cdot I_{\mathsf{com}}$ which, under Deuring's correspondence (Eq. (5)), is in bijection with $\mathrm{Hom}(E_{\mathsf{chl}}, E_{\mathsf{com}})$.
- It draws an element $\alpha_{\mathsf{rsp}} \in \Lambda$ of odd norm by rejection-sampling from the lattice-ball intersection $\Lambda \cap B(2^{e_{\mathsf{rsp}}-1})$, with $e_{\mathsf{rsp}} = 1 + \left\lceil \frac{1}{4}(\lambda + 2 \cdot \log_2(p)) \right\rceil$. The radius of the ball ensures that α_{rsp} exists and can be sampled efficiently.
- It evaluates the Deuring correspondence to compute $\varphi_{\mathsf{rsp}} = \Delta(\alpha_{\mathsf{rsp}})$. The signature is then the pair $(\mathsf{chl}, \varphi_{\mathsf{rsp}})$.

The last step, evaluating the Deuring correspondence, is arguably the most technical one. Indeed representing φ_{rsp} requires exiting the domain of elliptic curves and making a detour through "higher dimensions". The idea, originating in the SIDH attacks [CD23; MMP+23; Rob23], consists in embedding φ_{rsp} in a *push-out diagram*. It is sufficient, indeed, to find any other isogeny $\varphi_{\mathsf{aux}} : E_{\mathsf{chl}} \to E'_{\mathsf{aux}}$ to define a *2-dimensional isogeny* $\Phi_{\mathsf{rsp}} : E_{\mathsf{chl}} \times E_{\mathsf{aux}} \to E_{\mathsf{com}} \times E'_{\mathsf{aux}}$ between *abelian surfaces* of degree $\deg \varphi_{\mathsf{rsp}} + \deg \varphi_{\mathsf{aux}}$. Picking $\deg \varphi_{\mathsf{aux}} = 2^{e_{\mathsf{rsp}}} - \deg \varphi_{\mathsf{rsp}}$ lets the signer represent Φ_{rsp} as a walk of length e_{rsp} in a graph of 2-dimensional isogenies, which can be efficiently computed and transmitted. This technique is described in depth in Section 4.5.

We summarize in Section 2.3.2 the curves and isogenies computed, explicitly or implicitly, by the signing procedure.

2.3.3. Verification

Upon receiving a signature $(\mathsf{chl}, \varphi_{\mathsf{rsp}})$, after performing a few basic correctness checks (e.g.,, that all transmitted curves are supersingular), a verifier:

- Recomputes $K_{\mathsf{chl}} = P_{\mathsf{pk}} + [\mathsf{chl}]Q_{\mathsf{pk}}$ and derives $E_{\mathsf{chl}} = E_{\mathsf{pk}}/\langle K_{\mathsf{chl}} \rangle$;
- Checks that the domain of φ_{rsp} matches E_{chl} and evaluates the isogeny to obtain its codomain E'_{com};
- Recomputes the hash $H((E_{\mathsf{pk}}, P_{\mathsf{pk}}, Q_{\mathsf{pk}}), E'_{\mathsf{com}}, \mathsf{msg})$ and checks that its output matches chl.

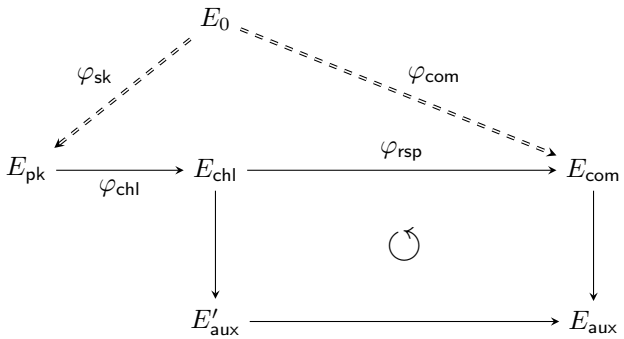


FIGURE 3. More detailed overview of the construction of a SQISIGN signature: To compute and transmit the response isogeny φ_{rsp} efficiently, the signer constructs a 2D isogeny $\Phi\colon E_{\mathsf{chl}} \times E'_{\mathsf{aux}} \to E_{\mathsf{com}} \times E_{\mathsf{aux}}$ from the bottom push-out thus embedding φ_{rsp}. The verifier receives compressed representations of φ_{chl} (thus E_{chl}), E_{aux}, and the kernel of Φ, allowing them to reconstruct Φ and thus recover E_{com} in order to check that φ_{chl} was honestly generated.

Algorithm 2.1 Side-by-side comparison of SQISIGN and Schnorr's signature

Parameters: p, E_0, O_0 as defined in this chapter, a security parameter λ

function KEY GENERATION
- Sample an $\mathcal{O}_0$-ideal I_{sk} of odd prime norm such that $\mathrm{cls}(I_{\mathsf{sk}})$ is uniformly distributed
- $E_{\mathsf{pk}}, \varphi_{\mathsf{sk}} \leftarrow \mathrm{cls}(I_{\mathsf{sk}}) \star E_0, \quad I_{\mathsf{sk}} \star E_0$
- Choose a basis $(P_{\mathsf{pk}}, Q_{\mathsf{pk}})$ of $E_{\mathsf{pk}}[2^\lambda]$
- **return** $(E_{\mathsf{pk}}, P_{\mathsf{pk}}, Q_{\mathsf{pk}})$, $(I_{\mathsf{sk}}, \varphi_{\mathsf{sk}})$

function SIGN($I_{\mathsf{sk}}, \varphi_{\mathsf{sk}}$, msg)
- Sample an $\mathcal{O}_0$-ideal I_{com} such that $\mathrm{cls}(I_{\mathsf{com}})$ is uniformly distributed
- $E_{\mathsf{com}} \leftarrow \mathrm{cls}(I_{\mathsf{com}}) \star E_0$
- $\mathsf{chl} \leftarrow H((E_{\mathsf{pk}}, P_{\mathsf{pk}}, Q_{\mathsf{pk}}) \,||\, E_{\mathsf{com}} \,||\, \mathsf{msg})$
- $I_{\mathsf{chl}} \leftarrow \{\omega \in \mathcal{O}_0 \mid \omega\widehat{\varphi_{\mathsf{sk}}}(P_{\mathsf{pk}} + [\mathsf{chl}]Q_{\mathsf{pk}}) = 0\}$
- $\Lambda \leftarrow \overline{I_{\mathsf{chl}} \cap I_{\mathsf{sk}}} \cdot I_{\mathsf{com}}$
- Sample α_{rsp} from Λ of odd norm $\mathrm{nrd}(\alpha_{\mathsf{rsp}}) < 2^{e_{\mathsf{rsp}}-1}$
- $\varphi_{\mathsf{rsp}} \leftarrow \Delta(\alpha_{\mathsf{rsp}})$
- **return** $(\mathsf{chl}, \varphi_{\mathsf{rsp}})$

function VERIFY($(E_{\mathsf{pk}}, P_{\mathsf{pk}}, Q_{\mathsf{pk}})$, $(\mathsf{chl}, \varphi_{\mathsf{rsp}})$, msg)
- $E'_{\mathsf{chl}} \leftarrow E_{\mathsf{pk}}/\langle P_{\mathsf{pk}} + [\mathsf{chl}]Q_{\mathsf{pk}}\rangle$
- Let $\varphi_{\mathsf{rsp}} : E_{\mathsf{chl}} \to E'_{\mathsf{com}}$
- $\mathsf{chl}' \leftarrow H((E_{\mathsf{pk}}, P_{\mathsf{pk}}, Q_{\mathsf{pk}}) \,||\, E'_{\mathsf{com}} \,||\, \mathsf{msg})$
- **return** $E'_{\mathsf{chl}} == E_{\mathsf{chl}}$ **and** $\mathsf{chl}' == \mathsf{chl}$

Parameters: An elliptic curve point G generating a group of prime order q

function KEY GENERATION
- Sample an integer $\mathsf{sk} \in [0, q-1]$
- **return** $A_{\mathsf{pk}} \leftarrow \mathsf{sk} \cdot G$

function SIGN(sk, msg)
- Sample an integer $\mathsf{com} \in [0, q-1]$
- $A_{\mathsf{com}} \leftarrow \mathsf{com} \cdot G$
- $\mathsf{chl} \leftarrow H(A_{\mathsf{pk}} \,||\, A_{\mathsf{com}} \,||\, \mathsf{msg})$
- $\sigma \leftarrow \mathsf{com} - \mathsf{chl} \cdot \mathsf{sk}$
- **return** (chl, σ)

function VERIFY(A_{pk}, (chl, σ), msg)
- $A'_{\mathsf{com}} \leftarrow \sigma \cdot G + \mathsf{chl} \cdot A_{\mathsf{pk}}$
- $\mathsf{chl}' \leftarrow H(A_{\mathsf{pk}} \,||\, A'_{\mathsf{com}} \,||\, \mathsf{msg})$
- **return** $\mathsf{chl}' == \mathsf{chl}$

CHAPTER 3

Protocol Specification

SQIsign is defined by two parameters: a *prime characteristic* p of the form $p = c \cdot 2^f - 1$, and a *security parameter* $\lambda \in \{128, 192, 256\}$. See Chapter 5 for a list of recommended parameters.

From the choice of p, follow the mathematical objects described in Section 2.2: the finite fields $\mathbb{F}_p$ and $\mathbb{F}_{p^2}$, the quaternion algebra $B_{p,\infty} = \mathbb{Q}\langle \mathbf{i}, \mathbf{j}, \mathbf{k} \rangle$, its maximal order $\mathcal{O}_0$, and the curve $E_0 \colon y^2 = x^3 + x$. We will use these symbols throughout the rest of the document without recalling their definitions again.

3.1. Data types

We list here the data types necessary for a compliant implementation of SQIsign.

3.1.1. Primitive data types

In the context of SQIsign, we call "primitive" the following three data types, though in most programming languages they would be considered derived.

Byte arrays (Bytes): a sequence of bytes of arbitrary length. Strings in quotes, e.g. `"sqisign"` are to be interpreted as the byte sequence of the ASCII encodings of the individual characters. All data types have a byte encoding described in Chapter 6.

Integer (Int): a signed integer with an arbitrary number of digits. When integers and derived types need to be converted from / to byte arrays, we always use two's complement little-endian encoding.

Finite-field element (FpElem): an element of the finite field $\mathbb{F}_p$, i.e. an integer modulo p. The characteristic p being fixed, most optimized implementations would have a dedicated implementation for FpElem, rather than building it on top of Int.

3.1.2. Composite data types

From the primitive types, SQIsign derives the following composite types.

$\mathbb{F}_{p^2}$ element (Fp2Elem): an element of the finite field $\mathbb{F}_{p^2} = \mathbb{F}_p[\mathrm{i}] = \mathbb{F}_p[I]/(I^2 + 1)$. These are represented by a pair $(a, b) \colon (\mathsf{FpElem})^2$, representing the field element $a + b\mathrm{i}$.

Elliptic curve (ECCurve): an elliptic curve defined over $\mathbb{F}_{p^2}$. All curves in SQIsign admit the Montgomery form

$$E_A \colon y^2 = x^3 + Ax^2 + x,$$

for some $A \colon \mathsf{Fp2Elem}$ with $A^2 \neq 4$. Accordingly, we represent an ECCurve by its A-coefficient.

Every Montgomery curve has at most 5 other isomorphic Montgomery curves. To ensure correctness, ECCurve must always use the largest among the 6 possible A for a given curve by the ordering Fp2Compare. See NormalizeMontgomery and ThetaToMontgomery.

For efficiency, implementations may use a projective representation $(A : C)$, corresponding to the A-coefficient A/C.

Elliptic curve point (ECPoint): a point on an ECCurve. This can be the special point at infinity or a point of coordinates $(x, y) \colon (\mathsf{Fp2Elem})^2$ satisfying the curve equation.

For efficiency, implementations may use projective x-only coordinates, i.e., pairs $(X : Z)$ such that $x = X/Z$, or the pair $(1 : 0)$ to represent the point at infinity. Though this representation does not distinguish a point (x, y) from its opposite $(x, -y)$, it is sufficient for use in SQIsign.

Elliptic curve torsion basis ($\mathsf{ECBasis}_e$): a 2^e-torsion basis on an ECCurve, i.e., a pair $(P, Q) \colon (\mathsf{ECPoint})^2$ of order 2^e such that any other 2^e-torsion point can be written as $[a]P + [b]Q$ for some $0 \leq a, b < 2^e$.

Implementations using x-only coordinates may represent a torsion basis (P, Q) by the triplet of x-coordinates (x_P, x_Q, x_{P-Q}): $(\mathsf{Fp2Elem})^3$. Though this representation does not distinguish (P, Q) from $(-P, -Q)$, it is sufficient for use in SQISIGN.

Torsion basis hint (Hint_e): as a bandwidth-saving measure, SQISIGN compresses some $\mathsf{ECBasis}_e$ down to a single byte, called a *hint*. A hint is essentially a counter in a torsion-basis-generation algorithm: by storing the counter instead of the basis's points, considerable space savings can be achieved for SQISIGN's public key and signature. For efficiency reasons, a hint only determines a basis up to sign: (P, Q) and $(-P, -Q)$ have the same hint; a hint can be decompressed to either basis without affecting results.

Integer matrix (Matrix): A 2×2 matrix with Int entries.

Quaternion (QuatElem): an element of $B_{p,\infty}$, i.e., an expression of the form

$$a + b\mathbf{i} + c\mathbf{j} + d\mathbf{k} \qquad a, b, c, d \in \mathbb{Q}\,.$$

A quaternion can be stored as a 4-tuple of rationals or, by collecting denominators, as a 5-tuple of Int.

Inert integral left $\mathcal{O}_0$-ideals (InertIdeal): full-rank sublattices of $\mathcal{O}_0$ stable by left multiplication by $\mathcal{O}_0$. When an $\mathcal{O}_0$-ideal I has a basis

$$\left\langle N, \quad N\mathbf{i}, \quad x + y\mathbf{i} + \frac{\mathbf{i}+\mathbf{j}}{2}, \quad -y + x\mathbf{i} + \frac{\mathbf{k}-1}{2} \right\rangle,$$

with N the norm of I and $0 \le x, y < N$, we say that I is *inert*. Inert ideals are represented by the nested tuple $(N, (x, y))$: $(\mathsf{Int})^3$.

Not every $\mathcal{O}_0$-ideal is inert, but every ideal class contains an inert ideal. See Section 4.3.

Integral left $\mathcal{O}_0$-ideals ($\mathsf{GenericO_0Ideal}$): For ideals that do not have an inert representation, we use the type $\mathsf{GenericO_0Ideal}$. Implementations are free to choose any representation for these. This specification represents them as $J = I\alpha$ where I is an inert ideal and α a quaternion. See Section 4.3.

3.1.3. Cryptographic types

Finally, public / private keys and signatures are represented by the following composite types.

Public key (PublicKey): A type consisting of:
- PublicKey.E_pk: An ECCurve;
- PublicKey.hint_pk: A $\mathsf{Hint}_{\lambda+2}$ defining a torsion basis on .E_pk.

Secret key (SecretKey): A type consisting of:
- SecretKey.pk: The PublicKey associated to this secret key;
- SecretKey.I_sk: A InertIdeal;
- SecretKey.M_sk: A Matrix encoding a torsion basis on .pk.E_pk;

Signature (Signature): A type consisting of:
- Signature.chl: The λ-bit challenge;
- Signature.E_aux: An ECCurve;
- Signature.M_chl: A Matrix encoding a torsion basis on .E_aux;
- Signature.hint_aux: A $\mathsf{Hint}_{e_{\mathsf{rsp}}+2}$ defining a torsion basis on .E_aux;
- Signature.hint_chl: A $\mathsf{Hint}_{e_{\mathsf{rsp}}+2}$ defining a torsion basis.

3.2. Constants

From p and λ, several derived constants are defined. We list them in Table 5 and treat them as global constants implicitly accessible to every subroutine. Further constants for localized use may be defined in the relevant sections.

The torsion basis $\mathcal{B}_0 = (P_0, Q_0)$ of $E_0[2^f]$ must satisfy $[2^{f-1}]Q_0 = (0, 0)$. Precomputed bases $\mathcal{B}_0$ for all SQISIGN parameters are given in Section 4.7.

TABLE 5. Global algorithmic constants

Names	Type	Description	See
p	Int	The prime characteristic	
λ	Int	The security parameter (must be a multiple of 8)	
f	Int	The exponent such that $p = c \cdot 2^f - 1$ with c odd	
e_{rsp}	Int	$e_{\mathsf{rsp}} = \lceil \log(p)/2 \rceil + \lambda/4 + 1$	Section 9.5
i	Fp2Elem	The standard generator of $\mathbb{F}_{p^2} \,/\, \mathbb{F}_p$	Section 4.2
E_0	ECCurve	The supersingular elliptic curve $y^2 = x^3 + x$ over $\mathbb{F}_{p^2}$	Section 4.4
$\mathcal{B}_0 = (P_0, Q_0)$	$\mathsf{ECBasis}_f$	A fixed torsion basis of $E_0[2^f]$	Section 4.7
$\mathbf{i}, \mathbf{j}, \mathbf{k} = \mathbf{ij}$	QuatElem	Algebra generators of $B_{p,\infty}$	Section 4.3

3.3. Symmetric cryptography

SQISIGN makes use of the SHAKE256 function both as a hash function for the Fiat–Shamir transform and as an extendable-output function for pseudorandom number generation. Though FIPS-202 [NIST15] defines a bit-oriented API for SHAKE256, we implicitly use a byte-oriented API for it, whereby passing a byte array str to

$$\mathsf{SHAKE256.Absorb}(\mathsf{ctx}, \mathsf{str})$$

implicitly converts it to a bit-string, and for any integer ℓ calling

$$\mathsf{SHAKE256.Squeeze}(\mathsf{ctx}, 8\ell)$$

returns an ℓ-byte array.

3.3.1. Pseudorandom bytes

SQISIGN uses stateful pseudorandom number generators (PRNG) to feed the randomized subroutines in key-generation and signing. We use an object-oriented syntax to represent the stateful PRNG: the data type PRNG is a wrapper around a SHAKE256 context, which it seeds and from which it squeezes pseudorandom bytes. Before use, a PRNG needs to be initialized by calling its .Init method, which seeds SHAKE256 with 48 bytes of secret random material, followed by a 3-byte ASCII code for domain separation. The domain separators used by SQISIGN are listed in Table 6.

TABLE 6. Domain separators in SQISIGN.

ASCII code	Use
`"key"`	Key generation: sampling secret keys
`"com"`	Signing: sampling commitments
`"res"`	Signing: sampling responses
`"def"`	Default: randomness in Las Vegas subroutines.[1] Substituting another (P)RNG for it produces the same outputs.

Algorithm 3.1 PRNG.Init(prng: PRNG, seed: Bytes, domain: Bytes)

Input: seed: Bytes a 48-byte seed
domain: Bytes a 3-byte domain separator

1: prng.ctx ← SHAKE256.Init()
2: prng.ctx ← SHAKE256.Absorb(ctx, seed)
3: prng.ctx ← SHAKE256.Absorb(ctx, domain)

After initialization, pseudorandom bytes are generated by calling the .Bytes method.

[1] Las Vegas algorithms are randomized algorithms that always produce the correct result.

Algorithm 3.2 PRNG.Bytes(prng : PRNG, ℓ : Int) : Bytes

Input: prng : PRNG
ℓ : Int a number of bytes to output
Output: ℓ pseudorandom bytes from the PRNG
1: prng.ctx, out $\leftarrow$ SHAKE256.Squeeze(prng.ctx, 8ℓ)
2: **return** out

3.3.2. Hash function

Following the Fiat–Shamir paradigm, signing in SQIsign hashes the public key, commitment and message to obtain a challenge. During verification, the same hash value is recomputed and checked against the signature. The hash function is defined by absorbing the binary encodings of a domain separator `"SQI"`, the public key, the commitment and the message in the SHAKE256 sponge, and then squeezing λ bits out.

Algorithm 3.3 HASH_λ(pk : PublicKey, E_{com} : ECCurve, msg : Bytes)

Input: pk : PublicKey a public key
E_{com} : ECCurve an elliptic curve
msg : Bytes an arbitrary-length byte array holding a message to sign
Output: A $(\lambda/8)$-byte array
1: ctx $\leftarrow$ SHAKE256.Init()
2: ctx $\leftarrow$ SHAKE256.Absorb(ctx, `"SQI"`) // 3 bytes (not including null terminator)
3: ctx $\leftarrow$ SHAKE256.Absorb(ctx, CurveToMontgomeryA(pk.E_pk))
4: ctx $\leftarrow$ SHAKE256.Absorb(ctx, pk.hint_pk)
5: ctx $\leftarrow$ SHAKE256.Absorb(ctx, CurveToMontgomeryA(E_{com}))
6: ctx $\leftarrow$ SHAKE256.Absorb(ctx, msg)
7: ctx, out $\leftarrow$ SHAKE256.Squeeze(ctx, λ)
8: **return** out

3.4. Algorithms

Assignment of a certain value α to a variable x is denoted by $x \leftarrow \alpha$. We use the **continue** statement to skip the remaining set of instructions inside a loop. Algorithms are typeset in sans serif font, e.g., IdealToIsogeny.

3.4.1. Exception handling

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

```
try
    Do something that might go wrong.
    if something is actually wrong then
        raise Exception ("Description of the error")   // Exit the try block and go to the except block
except
    Try to recover or show the user an error message.
```

3.4.2. Elementary subroutines

We specify here a few basic subroutines that operate on SQIsign's data types. Chapter 4 contains detailed descriptions of their implementation.

Algorithm 3.4 BytesToInt(bs: Bytes, ℓ: Int): Int

Takes as input a byte array bs of length ℓ and converts it to a non-negative integer in the range $\{0, \ldots, 256^\ell - 1\}$ using the little-endian convention.

Algorithm 3.5 CurveToMontgomeryA(E: ECCurve): Fp2Elem

Returns the Montgomery A-coefficient of the curve E.

Algorithm 3.6 BiscalarMul($\mathcal{B}$: ECBasis, m: Int, n: Int): ECPoint

If $\mathcal{B} = (P, Q)$ is a torsion basis, return the point $[m]P + [n]Q$. In x-only arithmetic, both input and output are only determined up to sign. See LadderBiscalar in Section 4.4.

Algorithm 3.7 ActOnBasis($\mathcal{B}$: ECBasis$_e$, $\mathbf{M}$: Matrix): ECBasis$_e$

Given a matrix $\mathbf{M} = \left(\begin{smallmatrix} x_1 & x_2 \\ x_3 & x_4 \end{smallmatrix}\right)$ and a torsion basis $\mathcal{B} = (P, Q)$, output the basis $\mathcal{B} \cdot \mathbf{M}$ as defined in Eq. (1), i.e.

$$(P, Q) \cdot \mathbf{M} = \big([x_1]P + [x_3]Q,\ [x_2]P + [x_4]Q \big).$$

In x-only coordinates, the inputs and output are determined up to sign.

Algorithm 3.8 ChangeOfBasis(E: ECCurve, e: Int, $\mathcal{B}_1$: ECBasis$_f$, $\mathcal{B}_2$: ECBasis$_e$): Matrix

An inverse to ActOnBasis: on input bases $\mathcal{B}_1$ of $E[2^f]$ and $\mathcal{B}_2$ of $E[2^e]$ with $e \leq f$, find a matrix $\mathbf{M}$ with entries in $[0 \,..\, 2^e[$ such that $\mathcal{B}_2 = \mathcal{B}_1 \cdot 2^{f-e}\mathbf{M}$.

Implementations using x-only arithmetic may only know $\mathcal{B}_1, \mathcal{B}_2$ up to sign, and thus their output $\mathbf{M}$ may only be determined up to sign. Returning either of $\mathbf{M}$ or $-\mathbf{M} \mod 2^e$ is acceptable.

Algorithm 3.9 SetChangeOfBasisMatrix

Input: E_1, E_2: ECCurve
$\mathcal{B}_1, \mathcal{B}_2$: ECBasis$_e$ with $\langle \mathcal{B}_1 \rangle = E_1[2^e]$ and $\langle \mathcal{B}_2 \rangle = E_2[2^e]$
e: Int
Output: $\mathbf{M}_2$: Matrix with positive entries $< 2^e$
hint$_1$, hint$_2$: Int

1: $\mathcal{B}'_1$, hint$_1$ $\leftarrow$ TorsionBasisToHint(E_1, f, e)
2: $\mathcal{B}'_2$, hint$_2$ $\leftarrow$ TorsionBasisToHint(E_2, f, e)
3: $\mathbf{M}_1 \leftarrow$ ChangeOfBasis($E_1, e, \mathcal{B}'_1, \mathcal{B}_1$)
4: $\mathcal{B}_2 \leftarrow$ ActOnBasis($\mathcal{B}_2, \mathbf{M}_1^{-1}$)
5: $\mathbf{M}_2 \leftarrow$ ChangeOfBasis($E_2, e, \mathcal{B}'_2, \mathcal{B}_2$)
6: **return** $\mathbf{M}_2$, hint$_1$, hint$_2$

Algorithm 3.10 NormalizeMatrix($\mathbf{M}$: Matrix, e: Int): Matrix

Input: $\mathbf{M}$: Matrix with entries in $[0 \,..\, 2^e[$
e: Int
Output: The lexicographically smallest among $\mathbf{M}$ and $-\mathbf{M} \mod 2^e$

1: Let $\mathbf{M} = \left(\begin{smallmatrix} m_1 & m_2 \\ m_3 & m_4 \end{smallmatrix}\right)$
2: **for** $i = 1$ to 4 **do**
3: **if** $0 < m_i < 2^{e-1}$ **then**
4: **return** $\mathbf{M}$
5: **else if** $m_i > 2^{e-1}$ **then**
6: **return** $-\mathbf{M} \mod 2^e$
7: **return** $\mathbf{M}$

3.4.3. Hints

Given a curve E, a *hints* is a one-byte representations of a basis of $E[2^e]$. Torsion bases of elliptic curves are, essentially, generated by incrementing a counter until certain equations are satisfied. A hint is the state of such counter. Reconstructing a basis from a hint can be much faster than regenerating one from scratch, as the states that do not yield a basis do not need to be tried.

SQIsign defines a function to generate bases along with a hint and one to decode a hint to a basis. Compliant implementations must implement these function exactly as they are specified here. See Section 4.4.2.

Algorithm 3.11 TorsionBasisToHint(E: ECCurve, e: Int, e_h: Int): (ECBasis$_e$, Hint$_{e-e_h}$)

On input a curve E and integers e, e_h, generate a basis $\mathcal{B} = (P, Q)$ of $E[2^e]$ and return a hint that decodes to the basis $([2^{e_h}]P, 2^{e_h}Q)$ of $E[2^{e-e_h}]$.

Algorithm 3.12 TorsionBasisFromHint(E: ECCurve, e: Int, hint: Hint$_e$): ECBasis$_e$

Decode a hint to a basis of $E[2^e]$, or fail if the hint does not define a valid torsion basis.

Optimized implementations may drop the validity check if they have an indirect mean of checking validity later. For example, the routine described in Section 4.4.2 always returns a torsion basis, though not necessarily of order 2^e. This failure can be detected shortly after calling TorsionBasisFromHint when computing scalar multiplication or isogeny chains, and is thus not checked explicitly. The existence of a basis of $E[2^e]$ can also be used as a test of supersingularity of the curve E.

3.4.4. Ideals

The following are routines related to quaternionic ideals.

Algorithm 3.13 IdealNorm(I: InertIdeal[2]): Int

Return the norm of the ideal I. the value N of the triple $N, (x, y)$ corresponding to I.

Algorithm 3.14 UniformO$_0$ClassSampling(M: Int, prng: PRNG): InertIdeal

Generate a random inert $\mathcal{O}_0$-ideal I of norm coprime to M, or of prime norm if $M = 0$, such that the statistical distance between $\mathrm{cls}(())I)$ and the uniform distribution on ideal classes is negligible in λ.

Compliant implementations must implement UniformO$_0$ClassSampling exactly as described in Section 4.3.

Algorithm 3.15 IdealToIsogeny(I: InertIdeal, prng: PRNG): (ECCurve, ECBasis$_f$)

On input an $\mathcal{O}_0$-ideal I of odd norm, output $E = \mathrm{cls}(())I) \star E_0$ and, letting $\varphi_I : E_0 \to E$, the basis $\big(\varphi_I(P_0), \varphi_I(Q_0)\big)$ of $E[2^f]$.

Algorithm 3.16 KernelDecomposedToIdeal(e: Int, c_1: Int, c_2: Int): GenericO$_0$Ideal

Given integers c_1, c_2, both divisible by 2^{f-e} but not both divisible by 2^{f-e+1}, return the unique $\mathcal{O}_0$-ideal I such that $E_0[I] = \langle [c_1]P_0 + [c_2]Q_0 \rangle$.

Algorithm 3.17 Intersect(I: GenericO$_0$Ideal, J: InertIdeal): InertIdeal

Compute the intersection $I \cap J$.

SampleQuaternionResponse realizes the quaternionic side of selecting a response in SQIsign: it receives as input ideals $I_{\mathsf{sk,chl}} = I_{\mathsf{chl}} \cap I_{\mathsf{sk}}$ and I_{com}, computes $\Lambda = \overline{I_{\mathsf{sk,chl}}} \cdot I_{\mathsf{com}}$, and samples a uniformly random quaternion $\alpha_{\mathsf{rsp}} \in \Lambda$ of odd norm smaller than $2^{e_{\mathsf{rsp}}-1}$.

[2] this extends naturally to other types of ideal introduced in Section 4.3

Algorithm 3.18 SampleQuaternionResponse

Input: $I_{\mathsf{sk,chl}}$: InertIdeal,
I_{com}: InertIdeal,
$\mathcal{B}_{\mathsf{com}}$: $\mathsf{ECBasis}_f$,
$\mathsf{prng}_{\mathsf{res}}$: PRNG the PRNG for the `"res"` domain,
$\mathsf{prng}_{\mathsf{def}}$: PRNG the PRNG for the default domain
Output: $I'_{\mathsf{sk,chl}}$: InertIdeal equivalent to the input ideal but of prime norm,
α_{rsp}: QuatElem an element of the lattice $\Lambda = \overline{I'_{\mathsf{sk,chl}}} \cdot I_{\mathsf{com}}$,
$\alpha_{\mathsf{sk,chl}}$: QuatElem an element of $I'_{\mathsf{sk,chl}}$
q_{rsp}: Int equal to $\mathsf{Norm}(\alpha_{\mathsf{rsp}})/(\mathsf{IdealNorm}(I_{\mathsf{com}}) \cdot \mathsf{IdealNorm}(I'_{\mathsf{sk,chl}}))$

$I'_{\mathsf{sk,chl}}, \alpha_{\mathsf{sk,chl}} \leftarrow \mathsf{EquivalentCoprimeIdeal}\,(I_{\mathsf{sk,chl}}, 0, \mathsf{BitLenSkChall}, \mathsf{prng}_{\mathsf{def}})$
$\alpha_{\mathsf{sk,chl}} \leftarrow \mathsf{Conjugate}(\alpha_{\mathsf{sk,chl}})$
if $\mathsf{GCD}(\mathsf{IdealNorm}(I_{\mathsf{com}}), \mathsf{IdealNorm}(I'_{\mathsf{sk,chl}})) \neq 1$ **then**
  raise Exception ("Commitment degree is not coprime to the sk,chall degree") // See Section 9.3.2
$q_{\mathsf{rsp}}, \alpha_{\mathsf{rsp}} \leftarrow \mathsf{RandomBoundedElement}(\mathsf{IdealMultiplication}(I'_{\mathsf{sk,chl}}, I_{\mathsf{com}}), 2^{e_{\mathsf{rsp}}-1}, \mathsf{prng}_{\mathsf{res}})$
return $I'_{\mathsf{sk,chl}}, \alpha_{\mathsf{rsp}}, \alpha_{\mathsf{sk,chl}}, q_{\mathsf{rsp}}$

3.4.5. Isogenies

The following routines are related to isogeny computations in dimension 1 and 2.

Algorithm 3.19 $\mathsf{TwoIsogenyChain}(P\colon \mathsf{ECPoint}, E\colon \mathsf{ECCurve}, e\colon \mathsf{Int} = [Q\colon \mathsf{ECPoint}]) : (\mathsf{ECCurve}, [\mathsf{ECPoint}])$

Given a point P on a curve E of order 2^{e+2}, compute $E/\langle 4P\rangle$ and evaluate the unique isogeny $\varphi : E \to E/\langle 4P\rangle$ at each of the points in the list `pts`.

Algorithm 3.20 $\mathsf{DimTwoKaniIsogeny}(E_1\colon \mathsf{ECCurve}, \mathcal{B}_1\colon \mathsf{ECBasis}_{e+2}, E_2\colon \mathsf{ECCurve}, \mathcal{B}_2\colon \mathsf{ECBasis}_{e+2}, e\colon \mathsf{Int}, \mathtt{pts}\colon [(\mathsf{ECPoint})^2]) : ((\mathsf{ECCurve})^2, [(\mathsf{ECPoint})^2], \mathsf{Boolean})$

Takes as input curves E_1, E_2 such that there exists a commutative diagram (also called a Kani diamond, whence the name)

$$\begin{array}{ccc} E_1 & \xrightarrow{\varphi_1} & E_3 \\ \downarrow{\scriptstyle \psi_2} & \circlearrowleft & \downarrow{\scriptstyle \varphi_2} \\ E_4 & \xrightarrow{\psi_1} & E_2 \end{array}$$

with $\deg(\varphi_1) = \deg(\psi_1) = q$ and $\deg(\varphi_2) = \deg(\psi_2) = 2^e - q$. Also takes as input a basis $\mathcal{B}_1$ of $E_1[2^{e+2}]$ and $\mathcal{B}_2 = \varphi_2 \circ \varphi_1(\mathcal{B}_1)$ and a list `pts` of points of $E_1 \times E_2$.

Using 2-dimensional isogeny formulas, it computes and returns E_3, E_4 and, for each (P_1, P_2) in `pts`, the point $\big(\varphi_1(P) + \widehat{\varphi}_2(P_2), \widehat{\psi}_1(P_2) - \psi_2(P_1)\big)$ of $E_3 \times E_4$. In case of invalid inputs, it returns `false`.

The HDResponse is the isogeny counterpart of SampleQuaternionResponse, it uses the effective Deuring correspondence to compute an HD representation of the isogeny $\varphi_{\mathsf{rsp}} = \Delta(\alpha_{\mathsf{rsp}})$.

Algorithm 3.21 HDResponse

Input: $I_{\mathsf{sk,chl}}$: InertIdeal,
α_{rsp} : QuatElem element of $\overline{I_{\mathsf{sk,chl}}}$ with $\mathsf{Norm}(\alpha_{\mathsf{rsp}}) < 2^{e_{\mathsf{rsp}}-1}$
$\alpha_{\mathsf{sk,chl}}$: QuatElem element of $I_{\mathsf{sk,chl}}$
q_{rsp} : Int
E_{com} : ECCurve
$\mathcal{B}_{\mathsf{com}}$: $\mathsf{ECBasis}_f$
$\mathsf{prng}_{\mathsf{res}}$: PRNG the PRNG for the `"res"` domain
$\mathsf{prng}_{\mathsf{def}}$: PRNG the PRNG for the default domain
Output: E_{aux} : ECCurve,
$\mathcal{B}_{\mathsf{aux}}$: $\mathsf{ECBasis}_{e_{\mathsf{rsp}}+2}$ of $E_{\mathsf{aux}}[2^{e_{\mathsf{rsp}}+2}]$,
E_{chl} : ECCurve,
$\mathcal{B}_{\mathsf{chl}}$: $\mathsf{ECBasis}_{e_{\mathsf{rsp}}+2}$ of $E_{\mathsf{chl}}[2^{e_{\mathsf{rsp}}+2}]$,

1: **if** $\mathsf{GCD}(2^{e_{\mathsf{rsp}}} - q_{\mathsf{rsp}}, \mathsf{Norm}(\alpha_{\mathsf{sk,chl}})) \neq 1$ **then**
2: **raise** Exception ("Auxiliary degree is not coprime to sk or sk,chall degree") // See Section 9.3.2
3: $\theta_{\mathsf{aux}}, I_{\mathsf{aux}} \leftarrow \mathsf{RandomIdealGivenNorm}(2^{e_{\mathsf{rsp}}} - q_{\mathsf{rsp}}, \alpha_{\mathsf{sk,chl}}, \mathsf{prng}_{\mathsf{res}})$
4: $E'_{\mathsf{aux}}, \mathcal{B}'_{\mathsf{aux}} \leftarrow \mathsf{IdealToIsogeny}(\mathsf{Intersect}((\theta_{\mathsf{aux}}, I_{\mathsf{aux}}), I_{\mathsf{sk,chl}}), \mathsf{prng}_{\mathsf{def}})$
5: $\mathbf{M}_{\mathsf{rsp}} \leftarrow \mathsf{EndomorphismToMatrix}(\alpha_{\mathsf{rsp}})$
6: $\mathbf{M}_{\mathsf{aux}} \leftarrow \mathsf{EndomorphismToMatrix}(\theta_{\mathsf{aux}})$
7: $\mu \leftarrow \mathsf{ModularInverse}(\mathsf{IdealNorm}(I_{\mathsf{sk,chl}}), 2^f)$
8: $\mathcal{B}'_{\mathsf{aux}} \leftarrow \mathsf{ActOnBasis}(\mathcal{B}'_{\mathsf{aux}}, \mu \cdot \mathbf{M}_{\mathsf{aux}}\mathbf{M}_{\mathsf{rsp}})$
9: $q_{\mathsf{inv}} \leftarrow \mathsf{ModularInverse}(q_{\mathsf{rsp}}, 2^{e+2})$
10: $(P_{\mathsf{com}}, Q_{\mathsf{com}}) \leftarrow [2^{f-e-2}]\mathcal{B}_{\mathsf{com}}$
11: $(P'_{\mathsf{aux}}, Q'_{\mathsf{aux}}) \leftarrow [q_{\mathsf{inv}}2^{f-e-2}]\mathcal{B}'_{\mathsf{aux}}$
12: $E_{\mathsf{aux}}, E_{\mathsf{chl}}, [\mathcal{B}_{\mathsf{aux}}, \mathcal{B}_{\mathsf{chl}}], \square$
$\leftarrow \mathsf{DimTwoKaniIsogeny}(E_{\mathsf{com}}, (P_{\mathsf{com}}, Q_{\mathsf{com}}), E'_{\mathsf{aux}}, (P'_{\mathsf{aux}}, Q'_{\mathsf{aux}}), e, [(P_{\mathsf{com}}, 0_{E_{\mathsf{com}}}), (Q_{\mathsf{com}}, 0_{E'_{\mathsf{aux}}})], \mathtt{false})$

3.4.6. Key Generation

Algorithm 3.22 $\mathsf{SQIsign.KeyGen}(\mathsf{seed}: \mathsf{Bytes}): (\mathsf{SecretKey}, \mathsf{PublicKey})$

Input: A 48-byte random seed
Output: sk : SecretKey,
pk : PublicKey

1: $\mathsf{prng}_{\mathsf{def}} \leftarrow \mathsf{PRNG.Init}(\mathsf{seed}, \mathtt{"def"})$
2: $\mathsf{prng}_{\mathsf{key}} \leftarrow \mathsf{PRNG.Init}(\mathsf{seed}, \mathtt{"key"})$
3: $I_{\mathsf{sk}} \leftarrow \mathsf{UniformO_0ClassSampling}(0, \mathsf{prng}_{\mathsf{key}})$ // Sample a random $\mathcal{O}_0$-ideal of odd prime norm
4: $E_{\mathsf{pk}}, \varphi_{\mathsf{sk}}(\mathcal{B}_0) \leftarrow \mathsf{IdealToIsogeny}(I_{\mathsf{sk}}, \mathsf{prng}_{\mathsf{def}})$ // $E_{\mathsf{pk}}, \varphi_{\mathsf{sk}} \leftarrow \mathrm{cls}(I_{\mathsf{sk}}) \star E_0, I \star E_0$
5: $\mathcal{B}_{\mathsf{pk}}, \mathsf{hint}_{\mathsf{pk}} \leftarrow \mathsf{TorsionBasisToHint}(E_{\mathsf{pk}}, \lambda + 2, 0)$ // Choose a basis of $E_{\mathsf{pk}}[2^\lambda]$
6: $\mathbf{M}_{\mathsf{sk}} \leftarrow \mathsf{ChangeOfBasis}(E_{\mathsf{pk}}, \lambda + 2, \varphi_{\mathsf{sk}}(\mathcal{B}_0), \mathcal{B}_{\mathsf{pk}})$
7: $\mathbf{M}_{\mathsf{sk}} \leftarrow \mathbf{M}_{\mathsf{sk}} \bmod 2^\lambda$
8: $\mathbf{M}_{\mathsf{sk}} \leftarrow \mathsf{NormalizeMatrix}(\mathbf{M}_{\mathsf{sk}}, \lambda)$
9: $\mathsf{pk} \leftarrow (E_{\mathsf{pk}}, \mathsf{hint}_{\mathsf{pk}})$
10: $\mathsf{sk} \leftarrow (\mathsf{pk}, I_{\mathsf{sk}}, \mathbf{M}_{\mathsf{sk}})$
11: **return** sk, pk

3.4.7. Signing

Algorithm 3.23 SQIsign.Sign(sk: SecretKey, msg: Bytes, seed: Bytes): Signature

Input: Secret signing key sk,
message msg $\in \{0,1\}^*$
48-byte random seed seed
Output: σ: Signature

$\mathsf{prng}_{\mathsf{def}} \leftarrow \mathsf{PRNG.Init}(\mathsf{seed}, \texttt{"def"})$
$\mathsf{prng}_{\mathsf{com}} \leftarrow \mathsf{PRNG.Init}(\mathsf{seed}, \texttt{"com"})$
$\mathsf{prng}_{\mathsf{res}} \leftarrow \mathsf{PRNG.Init}(\mathsf{seed}, \texttt{"res"})$
Parse sk as $(\mathsf{pk}, I_{\mathsf{sk}}, \mathbf{M}_{\mathsf{sk}})$

▷ ***Commitment*** ◁
$I_{\mathsf{com}} \leftarrow \mathsf{UniformO_0ClassSampling}(2, \mathsf{prng}_{\mathsf{com}})$ // Sample a random $\mathcal{O}_0$-ideal of odd norm
$E_{\mathsf{com}}, \mathcal{B}_{\mathsf{com}} \leftarrow \mathsf{IdealToIsogeny}(I_{\mathsf{com}}, \mathsf{prng}_{\mathsf{def}})$ // $E_{\mathsf{com}} \leftarrow \mathrm{cls}(I_{\mathsf{com}}) \star E_0$

▷ ***Challenge*** ◁
$\mathsf{chl} \leftarrow \mathsf{HASH}_\lambda(\mathsf{pk}, E_{\mathsf{com}}, \mathsf{msg})$

▷ ***Response*** ◁
$a_{\mathsf{chl}} \leftarrow \mathsf{BytesToInt}(\mathsf{chl}, \lambda/8)$
$(c_1, c_2) \leftarrow \mathbf{M}_{\mathsf{sk}} \cdot [1, a_{\mathsf{chl}}]^T$
$I_{\mathsf{chl}} \leftarrow \mathsf{KernelDecomposedToIdeal}(\lambda, c_1, c_2)$ // $I_{\mathsf{chl}} \leftarrow \{\omega \in \mathcal{O}_0 \mid \omega\widehat{\varphi_{\mathsf{sk}}}(P_{\mathsf{pk}} + [\mathsf{chl}]Q_{\mathsf{pk}}) = 0\}$
$I_{\mathsf{sk,chl}} \leftarrow \mathsf{Intersect}(I_{\mathsf{chl}}, I_{\mathsf{sk}})$ // $I_{\mathsf{sk,chl}} = I_{\mathsf{chl}} \cap I_{\mathsf{sk}}$
$I_{\mathsf{sk,chl}}, \alpha_{\mathsf{rsp}}, \alpha_{\mathsf{sk,chl}}, q_{\mathsf{rsp}} \leftarrow \mathsf{SampleQuaternionResponse}(I_{\mathsf{sk,chl}}, I_{\mathsf{com}}, \mathsf{prng}_{\mathsf{res}}, \mathsf{prng}_{\mathsf{def}})$ // $\alpha_{\mathsf{rsp}} \leftarrow\$ \overline{I_{\mathsf{sk,chl}}} \cdot I_{\mathsf{com}}$

▷ ***Response packaging*** ◁
$E_{\mathsf{aux}}, \mathcal{B}_{\mathsf{aux}}, E_{\mathsf{chl}}, \mathcal{B}_{\mathsf{chl}} \leftarrow \mathsf{HDResponse}(I_{\mathsf{sk,chl}}, \alpha_{\mathsf{rsp}}, \alpha_{\mathsf{sk,chl}}, q_{\mathsf{rsp}}, E_{\mathsf{com}}, \mathcal{B}_{\mathsf{com}}, \mathsf{prng}_{\mathsf{res}}, \mathsf{prng}_{\mathsf{def}})$ // $\varphi_{\mathsf{rsp}} \leftarrow \Delta(\alpha_{\mathsf{rsp}})$
$\mathbf{M}_{\mathsf{chl}}, \mathsf{hint}_{\mathsf{aux}}, \mathsf{hint}_{\mathsf{chl}} \leftarrow \mathsf{SetChangeOfBasisMatrix}(E_{\mathsf{aux}}, E_{\mathsf{chl}}, \mathcal{B}_{\mathsf{aux}}, \mathcal{B}_{\mathsf{chl}}, e_{\mathsf{rsp}} + 2)$
$\mathbf{M}_{\mathsf{chl}} \leftarrow \mathsf{NormalizeMatrix}(\mathbf{M}_{\mathsf{chl}}, e_{\mathsf{rsp}} + 2)$
return $\sigma \leftarrow (\mathsf{chl}, E_{\mathsf{aux}}, \mathbf{M}_{\mathsf{chl}}, \mathsf{hint}_{\mathsf{aux}}, \mathsf{hint}_{\mathsf{chl}})$

3.4.8. Verification

Algorithm 3.24 SQIsign.Verify(msg: Bytes, σ: Signature, pk: PublicKey): Boolean

Input: A message msg, signature σ and public key pk
Output: A boolean indicating whether the signature is valid

Parse pk as $(E_{\mathsf{pk}}, \mathsf{hint}_{\mathsf{pk}})$
Parse σ as $(\mathsf{chl}, E_{\mathsf{aux}}, \mathbf{M}_{\mathsf{chl}}, \mathsf{hint}_{\mathsf{aux}}, \mathsf{hint}_{\mathsf{chl}})$
Check that all the entries of $\mathbf{M}_{\mathsf{chl}}$ are in $[0\,..\,2^{e_{\mathsf{rsp}}+2}[$, otherwise **return** `false`
Check that $\mathbf{M}_{\mathsf{chl}} == \mathsf{NormalizeMatrix}(\mathbf{M}_{\mathsf{chl}}, e_{\mathsf{rsp}} + 2)$, otherwise **return** `false`
$\mathcal{B}_{\mathsf{pk}} \leftarrow \mathsf{TorsionBasisFromHint}(E_{\mathsf{pk}}, \lambda + 2, \mathsf{hint}_{\mathsf{pk}})$
$a_{\mathsf{chl}} \leftarrow \mathsf{BytesToInt}(\mathsf{chl}, \lambda/8)$
$K_{\mathsf{chl}} \leftarrow \mathsf{BiscalarMul}(\mathcal{B}_{\mathsf{pk}}, 1, a_{\mathsf{chl}})$ // $K_{\mathsf{chl}} \leftarrow P_{\mathsf{chl}} + [a_{\mathsf{chl}}]Q_{\mathsf{chl}}$ for $\mathcal{B}_{\mathsf{pk}} = (P_{\mathsf{chl}}, Q_{\mathsf{chl}})$
$E'_{\mathsf{chl}}, \square \leftarrow \mathsf{TwoIsogenyChain}(K_{\mathsf{chl}}, E_{\mathsf{pk}}, \lambda, \square)$ // $E'_{\mathsf{chl}} \leftarrow E_{\mathsf{pk}}/\langle[4](K_{\mathsf{chl}})\rangle$
$P_{\mathsf{aux}}, Q_{\mathsf{aux}}, \square \leftarrow \mathsf{TorsionBasisFromHint}(E_{\mathsf{aux}}, e_{\mathsf{rsp}} + 2, \mathsf{hint}_{\mathsf{aux}})$
$\mathcal{B}_{\mathsf{chl}} \leftarrow \mathsf{TorsionBasisFromHint}(E'_{\mathsf{chl}}, e_{\mathsf{rsp}} + 2, \mathsf{hint}_{\mathsf{chl}})$
$P_{\mathsf{chl}}, Q_{\mathsf{chl}}, \square \leftarrow \mathsf{ActOnBasis}(\mathcal{B}_{\mathsf{chl}}, \mathbf{M}_{\mathsf{chl}})$
$E'_{\mathsf{com}}, E, \square, \mathtt{valid} \leftarrow \mathsf{DimTwoKaniIsogeny}(E'_{\mathsf{chl}}, (P_{\mathsf{chl}}, Q_{\mathsf{chl}}), E_{\mathsf{aux}}, (P_{\mathsf{aux}}, Q_{\mathsf{aux}}), e_{\mathsf{rsp}}, \square, \mathtt{true})$
$\mathsf{chl}' \leftarrow \mathsf{HASH}_\lambda(\mathsf{pk}, E'_{\mathsf{com}}, \mathsf{msg})$
return `valid` **and** $\mathsf{chl} == \mathsf{chl}'$

3.5. Modules

This specification document, as well as the SQISIGN implementation, is organized in distinct modules. Each module provides the functionalities necessary to work with one type of objects or to compute the algorithms that make up SQISIGN.

Integers. One of the key data types SQISIGN manipulates is the integer type. This module provides the algorithms to compute with integers: it offers basic functionality for integer arithmetic (e.g. addition, multiplication, square root, etc.), as well as some more advanced methods, such as primality testing or (extended) GCD.

Finite fields. This module offers the functionality to work with the other primitive data type used in SQISIGN: finite field elements, i.e. elements of $\mathbb{F}_p$ and $\mathbb{F}_{p^2}$. The module provides algorithms to compute $\mathbb{F}_p$ arithmetic (e.g. addition, multiplication, squaring, etc.) and builds on top of that to extend the same algorithms to elements of $\mathbb{F}_{p^2}$.

Quaternions. Quaternions play a fundamental role in SQISIGN's key generation and signing procedures. This module exposes algorithms related to quaternions, ideals, and maximal orders.

Elliptic curves. This module offers the algorithms to perform arithmetic on elliptic curves to support the higher-level algorithms. In particular, this module works with x-only arithmetic for efficiency reasons. Building on top of these elliptic curve arithmetic algorithms, this module also provides functionalities such as torsion basis computation, pairings, and 2-isogeny computations.

Higher-dimensional isogenies. This module provides the algorithms to compute isogenies in dimension 2. More precisely, the main functionality of this module is the Isogeny22Chain algorithm, which computes $(2^e, 2^e)$-isogenies between products of elliptic curves.

Ideal-to-isogeny translation. This module introduces two main algorithms: IdealToIsogeny, which translates any left $\mathcal{O}_0$-ideal into the corresponding isogeny, and KernelDecomposedToIdeal, which translates an isogeny $E_0 \to E$ into its corresponding left $\mathcal{O}_0$-ideal.

Other modules. The SQISIGN specification document and implementation rely on other smaller modules to provide additional functionalities, such as providing precomputed data, implementing core cryptographic algorithms (e.g. SHAKE256), and packaging signature-level functionalities (key generation, signing, and verification).

CHAPTER 4

Supporting Algorithms

This chapter describes the lower-level algorithms used to realize the interfaces introduced in Chapter 2. We explain how the reference implementation carries out the main arithmetic, quaternion, elliptic-curve, pairing, isogeny and ideal operations efficiently. Some of the design choices described here are required for compatibility, while some others are implementation strategies that may be replaced by equivalent algorithms offering better performance.

We also report implementation properties that are important beyond their functional correctness. These include the representation of intermediate values, constant-time behavior when secret data may be involved, and clarification of the places where the current reference code is still variable time. Our main goal is to make the algorithms in this chapter precise enough to faithfully reproduce the SQIsign implementation.

4.1. Integers

This section specifies the integer operations used throughout SQIsign. We first give the mathematical interfaces for the basic arithmetic, modular, and number-theoretic operations. We then describe the fixed-capacity multi-precision representation used by the reference implementation, followed by the algorithms for primality testing and random sampling and the relevant constant-time considerations. The presentation separates the mathematical interfaces from implementation choices wherever possible.

4.1.1. Arithmetic interface

4.1.1.1. Basic operations. We use the following operations throughout the specification.

Algorithm 4.1 $\mathsf{Add}(\mathsf{Int},\mathsf{Int})\colon \mathsf{Int} \quad (a,b)\mapsto a+b$

Algorithm 4.2 $\mathsf{Sub}(\mathsf{Int},\mathsf{Int})\colon \mathsf{Int} \quad (a,b)\mapsto a-b$

Algorithm 4.3 $\mathsf{Neg}(\mathsf{Int})\colon \mathsf{Int} \quad a\mapsto -a$

Algorithm 4.4 $\mathsf{Abs}(\mathsf{Int})\colon \mathsf{Int} \quad a\mapsto |a|$

Algorithm 4.5 $\mathsf{Mul}(\mathsf{Int},\mathsf{Int})\colon \mathsf{Int} \quad (a,b)\mapsto ab$

Algorithm 4.6 $\mathsf{Compare}(\mathsf{Int},\mathsf{Int})\colon \{-1,0,1\} \quad (a,b)\mapsto \mathrm{sign}(a-b)$

Algorithm 4.7 $\mathsf{BitLength}(\mathsf{Int})\colon \mathbb{Z}_{\geq 0} \quad a\mapsto \min\{\ell\in\mathbb{Z}_{\geq 0}\mid |a|<2^{\ell}\}$

Algorithm 4.8 $\mathsf{ShiftLeft}(\mathsf{Int},\mathbb{Z}_{\geq 0})\colon \mathsf{Int} \quad (a,e)\mapsto a\cdot 2^{e}$

Algorithm 4.9 $\mathsf{ShiftRight}(\mathsf{Int},\mathbb{Z}_{\geq 0})\colon \mathsf{Int} \quad (a,e)\mapsto \mathrm{sign}(a)\,\lfloor |a|/2^{e}\rfloor$

Algorithm 4.10 $\mathsf{Mod2Exp}(\mathsf{Int},\mathbb{Z}_{\geq 0})\colon \mathsf{Int} \quad (a,e)\mapsto a \bmod 2^{e}$

Algorithm 4.11 $\mathsf{DyadicValuation}(\mathsf{Int}\setminus\{0\})\colon \mathbb{Z}_{\geq 0} \quad a\mapsto v_2(a)$

Addition and subtraction use single-word add-with-carry and subtract-with-borrow primitives, choosing among compiler-supported implementations at compile time. Multiplication uses plain schoolbook multiplication with double-width accumulators. The code handles signed values by conditionally negating the operands and the result when needed.

4.1.1.2. Division and modular arithmetic.

Algorithm 4.12 Div(Int, Int): (Int, Int)

$$(a, b) \mapsto (q, r) \quad \text{with} \quad a = qb + r, \qquad |r| < |b|$$

The implementation of Div uses variable-time Burnikel–Ziegler division [BZ98], with schoolbook division for smaller cases.

Algorithm 4.13 Mod(Int, Int): Int $\quad (a, m) \mapsto a \bmod |m|$

The implementation of Mod uses the same Burnikel–Ziegler division with schoolbook fallback as Div, retaining only the remainder and returning the canonical representative modulo $|m|$.

Algorithm 4.14 Divides(Int, Int): Boolean $\quad (a, b) \mapsto (a \equiv 0 \bmod b)$

The implementation of Divides tests divisibility by computing the remainder and checking whether it is zero.

Algorithm 4.15 ModularInverse(Int, Int): Int $\cup \{\perp\}$ $\quad (a, m) \mapsto a^{-1} \bmod m$

The implementation of ModularInverse uses a Lehmer-based extended Euclidean algorithm to compute the inverse when it exists.

Algorithm 4.16 PowMod(Int, Int, Int): Int $\quad (a, e, m) \mapsto a^e \bmod m$

The exponentiation-based routines do not operate directly on the two's-complement limbs. They instead convert their operands to the separate `modqx` backend and use unsaturated Montgomery arithmetic. This backend uses radix 2^{61} on 64-bit systems and radix 2^{28} on 32-bit systems. It is a distinct multi-precision layer optimized for Montgomery multiplication and squaring and was inspired by Mike Scott's `modarith` arithmetic generator [Sco24]. It is used by `ibz_pow_mod`, modular square roots and primality testing.

4.1.1.3. Number-theoretic operations.

Algorithm 4.17 GCD(Int, Int): Int $\quad (a, b) \mapsto \gcd(a, b)$

The implementation of GCD uses a variable-time, double-word variant of Lehmer's algorithm [Leh38], which accelerates the Euclidean algorithm by performing several quotient steps using the most significant limbs of the inputs.

Algorithm 4.18 XGCD(Int, Int): (Int, Int, Int)

$$(a, b) \mapsto (g, u, v) \quad \text{with} \quad g = \gcd(a, b), \qquad ua + vb = g$$

The implementation of XGCD uses the same Lehmer-based approach while tracking the Bézout coefficients u and v such that $ua + vb = \gcd(a, b)$.

Algorithm 4.19 CRT(Int, Int, Int, Int): Int

$$(a_1, m_1, a_2, m_2) \mapsto x, \qquad x \equiv a_1 \pmod{m_1}, \quad x \equiv a_2 \pmod{m_2}$$

The implementation of CRT computes the inverse of m_1 modulo m_2 to combine the two congruences and reduces the result to the canonical representative $0 \le x < m_1 m_2$.

Algorithm 4.20 ModularSQRT(Int, Int): Int $\cup \{\perp\}$

$$(a, p) \mapsto x \quad \text{with} \quad x^2 \equiv a \pmod{p}$$

For an odd prime p, the implementation of ModularSQRT selects the method according to $p \bmod 4$. If $p \equiv 3 \pmod 4$, it computes the square root by exponentiation to $(p+1)/4$; if $p \equiv 1 \pmod 4$, it uses the Tonelli–Shanks algorithm.

Algorithm 4.21 SqrtFloor(Int): Int $\quad a \mapsto \lfloor \sqrt{a} \rfloor$

The implementation of SqrtFloor uses a division-free Newton iteration for the reciprocal square root of a normalized input. Starting from a low-precision approximation, it repeatedly applies $y \leftarrow \frac{y(3-\alpha y^2)}{2}$ at increasing precision. The resulting approximation is multiplied by the input and corrected exactly to obtain $\lfloor \sqrt{a} \rfloor$.

Algorithm 4.22 Legendre(Int, Int): $\{-1, 0, 1\}$ $\quad (a, p) \mapsto \left(\frac{a}{p}\right)$

The implementation computes the Legendre symbol using quadratic reciprocity laws. This is done by evaluating the Jacobi symbol by repeatedly removing factors of 2, applying the corresponding sign rule modulo 8,

and swapping the odd arguments using quadratic reciprocity. Since p is prime, the resulting Jacobi symbol is the Legendre symbol.

4.1.2. Reference integer representation and bounds

Definition 4.1.1. The type Int is represented by an `ibz_t`. The type `digit_t` is a 32- or 64-bit word depending on the target platform. An `ibz_t` contains a fixed array of `IBZ_NLIMBS` such words, together with a field `bitlen`. The constant `IBZ_NLIMBS` is level-dependent and belongs to the reference implementation. This constant fixes the available maximum size, but it does not change the integer interface. The limbs store a signed integer in two's-complement form. If $b = \texttt{bitlen}$, the stored value a satisfies $|a| < 2^{b-1}$.

Every `ibz_t` reserves its own limb array, which does not grow or shrink during a computation. The field `bitlen` gives a safe upper bound on the portion of the limb array that an operation may need to process, allowing unused limbs to be skipped.

We use `bitlen` as an upper bound on the value's length, not as its exact length. Recomputing the exact size after every operation would require inspecting the value and would make the working length depend on the data. Instead, we propagate size information through the arithmetic. Addition and subtraction use the larger input `bitlen` plus one bit, while multiplication uses the sum of the two input values minus one. If the initial `bitlen` values depend only on public parameters, the number of processed limbs does not depend on secret data. This does not make the integer layer constant time, but gives the basic multi-word arithmetic a useful fixed-shape structure.

As a result, a bound can become looser than the actual value after several operations. For a variable `n`, `n->bitlen` stores this safe bound, while `ibz_bitsize(n)` scans the integer and computes its actual bit length. An explicit bound is normally needed only when initializing an `ibz_t` from a fixed value or when a tighter public bound is known. The function `ibz_set_bound` performs this adjustment. Tightening a bound from secret-dependent information could make the number of active limbs depend on the secret.

This distinction also matters for primality testing, which is currently variable time. Montgomery arithmetic chooses the number of working words from `bitlen`. The function `ibz_probab_prime` computes the actual size s of a positive input with `ibz_bitsize(n)` and, when necessary, makes a local copy with upper bound $s+1$, where the extra bit is the sign bit. The input is not modified. This costs one size scan and (if needed) one copy, allowing the more expensive modular arithmetic to operate on fewer words.

4.1.3. Primality testing

Primality testing appears frequently in SQIsign. Key generation and signing generate many prime candidates, while Qlapoti uses primality testing inside its norm-equation search. The current implementation is variable time and tries to reject composite values as cheaply as possible before using modular exponentiation.

After handling $n < 2$, the primes 2 and 3, and even values, we apply two inexpensive sieve stages. In the 64-bit implementation, the first stage tests the primes from 3 to 53. We group them into two products smaller than 2^{32} and reduce n modulo each product rather than separately modulo every prime. If the candidate survives, the second sieve stage handles the remaining small primes by grouping them into fixed products smaller than 2^{64}. A single multi-precision reduction modulo one such product tests divisibility by several primes at once.

Since the sieve moduli are fixed, we precompute reciprocal information for each of them. This allows the reductions to replace generic division with multiplications, shifts, and a small correction, using the same basic idea as Barrett reduction. Most of the precomputed table is shared by all parameter sets, while the sieve extends to 509 or 1021 as required by each level. On platforms where this optimized reduction is not available, we use the same batching strategy but construct at run time the largest products of consecutive small primes that fit in one machine word.

Candidates that survive the sieves enter the probable-prime test. We build one Montgomery context for the candidate and reuse it throughout the test. We first perform a Miller–Rabin round with base 2 and then a strong Lucas–Selfridge test. These two tests form the Baillie–PSW part of the routine. Any failure immediately rejects the candidate.

The implementation also receives the parameter `reps`, which is level-dependent in its current state. After the Baillie–PSW test it performs $\max(\texttt{reps} - 24, 0)$ additional Miller–Rabin rounds with random witnesses in $[2, n-2]$. The current code uses $\texttt{reps} = 64, 96, 128$ at NIST levels I, III and V, giving at most 40, 72 and 104 additional rounds. We precompute R^2 once before these rounds and use it to convert each witness to Montgomery representation.

Algorithm 4.23 IsPrime(n)

Input: n : Int and `reps` : Int
Output: `true` if n passes the probable-prime test, `false` otherwise

if $n < 2$ **then**
  return `false`
if $n = 2$ **or** $n = 3$ **then**
  return `true`
if n is even **then**
  return `false`
if the first sieve stage rejects n **then**
  return `false`
if the second sieve stage rejects n **then**
  return `false`
if Miller–Rabin to base 2 rejects n **then**
  return `false`
if the strong Lucas–Selfridge test rejects n **then**
  return `false`
$t \leftarrow \max(\texttt{reps} - 24, 0)$
Precompute R^2 for Montgomery conversion
for i **from** 1 **up to** t **do**
  Sample a random witness $a \in [2, n-2]$
  if Miller–Rabin to base a rejects n **then**
    return `false`
return `true`

Before primality testing, `ibz_probab_prime` tightens the working bound of a local copy when necessary, as described above, allowing the Montgomery arithmetic to operate on fewer words.

Qlapoti also exploits the special form of the candidates that survive the primality test. The subsequent sum-of-two-squares computation needs only a square root of -1 modulo z, rather than the square root of an arbitrary value. We exploit this special case instead of using the general modular square-root routine. We first find a small prime a such that $\left(\frac{a}{z}\right) = -1$. For prime z, Euler's criterion gives

$$\left(a^{(z-1)/4}\right)^2 = a^{(z-1)/2} \equiv -1 \pmod{z}.$$

Hence a single modular exponentiation $r = a^{(z-1)/4} \bmod z$ gives the required square root of -1. Searching for a is inexpensive because we try small primes, and we verify $r^2 \equiv -1 \pmod{z}$ before using the result. The verified root is then passed directly to the sum-of-two-squares computation.

4.1.4. Random sampling

We also include a function for sampling a uniform pseudorandom integer in an interval $[a, b]$, with $b \geq a$. Let $m = b - a + 1$ and $B = \texttt{ibz\_bitsize}(\texttt{m})$. The implementation samples $B + 64$ bits and reduces the sampled value modulo m, giving at least 64 extra bits relative to the interval width.

Algorithm 4.24 SampleFromInterval$(a, b, \mathsf{prng}_{\mathsf{def}})$

Input: a, b : Int with $b \geq 0$ and $-b \leq a \leq b$, and $\mathsf{prng}_{\mathsf{def}}$: PRNG
Output: A pseudorandom integer in $[a, b]$

$m \leftarrow b - a + 1$
$n \leftarrow \texttt{ibz\_bitsize}(m)$
repeat
  Sample $\lceil n/8 \rceil$ bytes from $\mathsf{prng}_{\mathsf{def}}$
  Mask the unused high bits to obtain an n-bit positive integer x
until $x < m \cdot \lceil 2^n / m \rceil$
return $a + (x \bmod m)$

The rejection sampling ensures that x is drawn from an interval of size a full multiple of m, so there is no bias in the sampling, while the excess 64 bits make it overwhelmingly likely that the process succeeds on the first iteration.

4.1.5. Constant-time considerations

The MP module is not constant time as a whole. The fixed-capacity representation and public `bitlen` bounds are nevertheless useful steps towards that goal. They let the core multi-word routines execute over a publicly determined number of words without first scanning the value. Addition, subtraction, multiplication, negation, conditional negation and conditional swap already follow this pattern. The zero, one, sign and parity tests also use only the active limbs or the relevant boundary word.

On the other hand, several important routines still have value-dependent control flow. `ibz_cmp` returns when it finds the first differing limb and `ibz_bitsize` stops when it finds the highest non-sign limb. Division and reduction, dyadic valuation, Lehmer GCD/XGCD and modular inverse, CRT, sliding-window exponentiation and the modular square-root routines are also variable time. Primality testing is variable time as well.

The current representation therefore facilitates future constant-time improvements but is not constant-time at present.

4.2. Finite fields

This section specifies the finite-field operations used throughout SQIsign. We work over $\mathbb{F}_p$ and its quadratic extension $\mathbb{F}_{p^2}$, and require basic field arithmetic, quadratic residuosity and square roots, deterministic comparison, batch inversion, and discrete logarithms in 2-power subgroups.

Several algorithms require a deterministic comparison of elements of $\mathbb{F}_{p^2}$, in particular when making canonical normalization choices. We therefore define the following strict total order over $\mathbb{F}_{p^2}$, which depends on the chosen representation of its elements.

Definition 4.2.1. Let $x, y \in \mathbb{F}_{p^2}$, written as $x = x_0 + \mathsf{i}x_1$ and $y = y_0 + \mathsf{i}y_1$, with $x_0, x_1, y_0, y_1 \in \mathbb{F}_p$ represented by integers in $\{0, \dots, p-1\}$. Then

$$x < y \iff x_1 < y_1 \quad \text{or} \quad (x_1 = y_1 \text{ and } x_0 < y_0).$$

Algorithm 4.25 Fp2Compare(Fp2Elem, Fp2Elem) : Boolean $(x, y) \mapsto (x < y)$, where $<$ is the order of Definition 4.2.1

4.2.1. Field arithmetic

Recall that

$$\mathbb{F}_p = \mathbb{Z}/p\mathbb{Z}, \qquad \mathbb{F}_{p^2} = \mathbb{F}_p[\mathsf{i}]/(\mathsf{i}^2 + 1),$$

where $p \equiv 3 \pmod 4$. We use the following elementary operations for $x = x_0 + \mathsf{i}x_1$ and $y = y_0 + \mathsf{i}y_1$.

Algorithm 4.26 $\mathsf{Add}_{\mathbb{F}_{p^2}}$(Fp2Elem, Fp2Elem) : Fp2Elem $(x, y) \mapsto (x_0 + y_0) + \mathsf{i}(x_1 + y_1)$

Algorithm 4.27 $\mathsf{Sub}_{\mathbb{F}_{p^2}}$(Fp2Elem, Fp2Elem) : Fp2Elem $(x, y) \mapsto (x_0 - y_0) + \mathsf{i}(x_1 - y_1)$

Algorithm 4.28 $\mathsf{Neg}_{\mathbb{F}_{p^2}}$(Fp2Elem) : Fp2Elem $x \mapsto -x_0 - \mathsf{i}x_1$

Algorithm 4.29 $\mathsf{Mul}_{\mathbb{F}_{p^2}}$(Fp2Elem, Fp2Elem) : Fp2Elem $(x, y) \mapsto (x_0y_0 - x_1y_1) + \mathsf{i}(x_0y_1 + x_1y_0)$

Algorithm 4.30 $\mathsf{Sqr}_{\mathbb{F}_{p^2}}$(Fp2Elem) : Fp2Elem $x \mapsto (x_0^2 - x_1^2) + \mathsf{i}(2x_0x_1)$

Algorithm 4.31 $\mathsf{Inverse}_{\mathbb{F}_{p^2}}$(Fp2Elem) : Fp2Elem $x \mapsto (x_0 - \mathsf{i}x_1)(x_0^2 + x_1^2)^{-1}$

Algorithm 4.32 $\mathsf{Half}_{\mathbb{F}_{p^2}}$(Fp2Elem) : Fp2Elem $x_0 + \mathsf{i}x_1 \mapsto x_0/2 + \mathsf{i}(x_1/2)$

Algorithm 4.33 $\mathsf{Frobenius}_{\mathbb{F}_{p^2}}$(Fp2Elem) : Fp2Elem $x_0 + \mathsf{i}x_1 \mapsto x_0 - \mathsf{i}x_1$

4.2.1.1. Reference implementation. Unless stated otherwise, the finite-field layer implements these operations in constant time. The API follows the same rule. The explicit exception in the reference $\mathbb{F}_{p^2}$ code is `fp2_pow_vartime`, which uses ordinary square-and-multiply and must only be used when the exponent may be treated as public.

The reference $\mathbb{F}_{p^2}$ multiplication uses three multiplications in $\mathbb{F}_p$ by computing $(x_0 + x_1)(y_0 + y_1)$ together with x_0y_0 and x_1y_1. Squaring uses two multiplications in $\mathbb{F}_p$. The Frobenius map is just conjugation because $p \equiv 3 \pmod 4$.

4.2.1.1. Arithmetic in $\mathbb{F}_p$. The implementation uses Montgomery arithmetic and exploits the shape

$$p = c2^f - 1,$$

where c is small. The portable 64-bit implementation uses an unsaturated radix representation, while optimized assembly implementations may use a saturated full-word representation.

Since $p \equiv 3 \pmod 4$, inversion, quadratic residuosity and square roots reuse the fixed exponentiation

$$h(a) = a^{(p-3)/4}.$$

For $a \neq 0$,

$$\begin{aligned} a^{-1} &= ah(a)^4, \\ a^{(p-1)/2} &= ah(a)^2, \\ \sqrt{a} &= ah(a) \end{aligned}$$

when a is a quadratic residue. Each parameter set evaluates $h(a)$ with a fixed addition chain.

4.2.1.2. Quadratic residuosity in $\mathbb{F}_{p^2}$. For $x = x_0 + \mathrm{i}x_1$, let

$$N(x) = x_0^2 + x_1^2 \in \mathbb{F}_p\,.$$

Then x is a square in $\mathbb{F}_{p^2}$ if and only if $N(x)$ is a square in $\mathbb{F}_p$.

Algorithm 4.34 $\mathsf{IsSquare}_{\mathbb{F}_{p^2}}(\mathsf{Fp2Elem})\colon \mathsf{Boolean} \quad x_0 + \mathrm{i}x_1 \mapsto \left(x_0^2 + x_1^2 \text{ is a square in } \mathbb{F}_p\right)$

Algorithm 4.35 $\mathsf{Fp2SQRT}(x)$

Input: $x = x_0 + \mathrm{i}x_1 \in \mathbb{F}_{p^2}$, a quadratic residue
Output: $r \in \mathbb{F}_{p^2}$ such that $r^2 = x$

$\delta \leftarrow \mathsf{ModularSQRT}(x_0^2 + x_1^2, p)$
if $x_1 = 0$ **then**
  $\delta \leftarrow x_0$
$r_0 \leftarrow x_0 + \delta$
$t \leftarrow 2r_0$
$r_1 \leftarrow t^{(p-3)/4}$
$r_0 \leftarrow r_0r_1$
$r_1 \leftarrow x_1r_1$
if $(2r_0)^2 = t$ **then**
  return $r_0 + \mathrm{i}r_1$
else
  return $r_1 - \mathrm{i}r_0$

4.2.1.3. Square roots in $\mathbb{F}_{p^2}$. The reference implementation uses `fp_select`, a constant-time conditional selection primitive, both to handle the $x_1 = 0$ case and to select the returned square root, avoiding branches on value-dependent conditions. A separate `fp2_sqrt_verify` function squares the returned value and checks it against the input when verification is required.

4.2.1.4. Batch inversion. When several inversions are required at the same time, we use Montgomery's batch-inversion trick, as described in $\mathsf{Fp2BatchInverse}$. This algorithm uses one inversion and $3(n-1)$ multiplications.

4.2.1.5. Comparison and conditional operations. The order in Definition 4.2.1 is implemented by `fp2_less_than`. It first encodes both components canonically and then scans all bytes from most to least significant. Masked comparisons record the first difference without an early return. The functions `fp2_select` and `fp2_cswap` apply the corresponding constant-time operations independently to the real and imaginary parts.

Algorithm 4.36 Fp2BatchInverse$(x_1, \ldots, x_n)$

Input: $x_1, \ldots, x_n \in \mathbb{F}_{p^2}^{\times}$
Output: $(x_1^{-1}, \ldots, x_n^{-1})$
1: $c_1 \leftarrow x_1$
2: **for** j **from** 2 **up to** n **do**
3: $\quad c_j \leftarrow c_{j-1} x_j$
4: $t \leftarrow c_n^{-1}$
5: **for** j **from** n **down to** 2 **do**
6: $\quad y_j \leftarrow t c_{j-1}$
7: $\quad t \leftarrow t x_j$
8: $y_1 \leftarrow t$
9: **return** $(y_1, \ldots, y_n)$

4.2.2. Discrete logarithms

Pairings reduce the change-of-basis computation to discrete logarithms in $\mu_{2^e} \subset \mathbb{F}_{p^2}^{\times}$. We solve them with a recursive Pohlig–Hellman decomposition specialized to powers of two, computing the integer $a \in [0, 2^e)$ such that $f = g^a$.

Algorithm 4.37 Normalized DLog NormalizedDlog(f, g^{-1}, e)

Input: $f, g \in \mu_{2^e}$, g of order 2^e, and $f \in \langle g \rangle$
Output: $a \in \mathbb{Z}$, $0 \le a < 2^e$, such that $f = g^a$
1: **if** $e = 0$ **then**
2: $\quad$ **return** 0
3: **if** $e = 1$ **then**
4: $\quad$ **return** 0 if $f = 1$, and 1 otherwise
5: $r \leftarrow \lfloor e/2 \rfloor$
6: $\ell \leftarrow e - r$
7: $a_0 \leftarrow$ NormalizedDlog$(f^{2^\ell}, (g^{-1})^{2^\ell}, r)$
8: $f \leftarrow f (g^{-1})^{a_0}$
9: $g^{-1} \leftarrow (g^{-1})^{2^r}$
10: $a_1 \leftarrow$ NormalizedDlog(f, g^{-1}, ℓ)
11: **return** $a_0 + 2^r a_1$

The C routine receives g^{-1} together with f. The implementation in `fp2_dlog_2e_rec` performs the same updates one binary digit at a time. When it finds a digit equal to one, it multiplies the pending value by the current inverse generator and squares that generator for the next digit. The two recursive halves are combined as $a_0 + 2^r a_1$.

The routine `ec_dlog_2_tate` applies this field DLog to change bases on the 2-power torsion. It assumes a full basis (P, Q) and a possibly smaller basis (R, S). It computes one reference Tate pairing and the four pairings needed for the coordinates, reduces and normalizes them together, and calls `fp2_dlog_2e` four times. The outputs must satisfy

$$R = [r_1]P + [r_2]Q, \qquad S = [s_1]P + [s_2]Q.$$

4.2.3. Implementation notes

The $\mathbb{F}_{p^2}$ layer is on purpose thin and builds most operations from the prime-field layer. Inversion uses one inversion in $\mathbb{F}_p$ through the norm $x_0^2 + x_1^2$, quadratic residuosity tests the same norm, and the Frobenius map only negates the imaginary component. As mentioned before, batch inversion reduces several $\mathbb{F}_{p^2}$ inversions at the price of one inversion and $3(n-1)$ multiplications.

The concrete $\mathbb{F}_p$ representation depends on the target architecture. The reference implementation uses Montgomery arithmetic, and more optimized targets may replace the low-level field operations without changing this interface.

4.3. Quaternions

In this section, we describe the supporting algorithms required in SQIsign for manipulating quaternion elements and ideals. The algorithms below assume a number of fixed and implicit parameters which are defined in Section 2.2:

- A prime number p equal to $3 \bmod 4$
- The quaternion algebra $B_{p,\infty} = \mathbb{Q}\langle 1, \mathbf{i}, \mathbf{j}, \mathbf{k}\rangle$
- The maximal order $\mathcal{O}_0 = \mathbb{Z}\langle 1, \mathbf{i}, \frac{\mathbf{i}+\mathbf{j}}{2}, \frac{1+\mathbf{k}}{2}\rangle \subset B_{p,\infty}$

4.3.1. Quaternion elements

This section deals with quaternion elements in $B_{p,\infty}$. Below, we introduce several functions to handle either generic elements in $B_{p,\infty}$ or elements in $\mathcal{O}_0$.

Definition 4.3.1. The type QuatElem consists of a tuple a, b, c, d, e of five Int values that represents $\frac{a+\mathbf{i}b+\mathbf{j}c+\mathbf{k}d}{e} \in B_{p,\infty}$.

4.3.1.1. Generic operations in $B_{p,\infty}$.

Algorithm 4.38 Conjugate(QuatElem) : QuatElem

$$\left(\frac{a+\mathbf{i}b+\mathbf{j}c+\mathbf{k}d}{e}\right) \mapsto \frac{a-\mathbf{i}b-\mathbf{j}c-\mathbf{k}d}{e}$$

Algorithm 4.39 Norm(QuatElem) : Int

$$\left(\frac{a+\mathbf{i}b+\mathbf{j}c+\mathbf{k}d}{e}\right) \mapsto \frac{a^2+b^2+p(c^2+d^2)}{e^2}$$

Algorithm 4.40 Trace(QuatElem) : Int

$$\left(\frac{a+\mathbf{i}b+\mathbf{j}c+\mathbf{k}d}{e}\right) \mapsto \frac{2a}{e}$$

Algorithm 4.41 CoordinatesB(QuatElem) : (Int, Int, Int, Int, Int)

$$\left(\frac{a+\mathbf{i}b+\mathbf{j}c+\mathbf{k}d}{e}\right) \mapsto \left(\frac{a}{g}, \frac{b}{g}, \frac{c}{g}, \frac{d}{g}, \frac{e}{g}\right) \quad \text{with } g = \mathsf{GCD}(a, b, c, d, e)$$

4.3.1.2. Special order $\mathcal{O}_0$, generic operations.

Algorithm 4.43 CoordinatesO$_0$(QuatElem) : Int, Int, Int, Int

$$\left(a + b\mathbf{i} + c\frac{\mathbf{i}+\mathbf{j}}{2} + d\frac{1+\mathbf{k}}{2}\right) \mapsto a, b, c, d$$

Algorithm 4.44 ModO$_0$(QuatElem, Int) : QuatElem

$$\left(a + b\mathbf{i} + c\frac{\mathbf{i}+\mathbf{j}}{2} + d\frac{1+\mathbf{k}}{2}, N\right) \mapsto (a \bmod N) + (b \bmod N)\mathbf{i} + (c \bmod N)\frac{\mathbf{i}+\mathbf{j}}{2} + (d \bmod N)\frac{1+\mathbf{k}}{2}$$

Algorithm 4.45 MakePrimitive(QuatElem) : QuatElem

$$\left(a + b\mathbf{i} + c\frac{\mathbf{i}+\mathbf{j}}{2} + d\frac{1+\mathbf{k}}{2}\right) \mapsto \frac{a}{g} + \frac{b}{g}\mathbf{i} + \frac{c}{g}\frac{\mathbf{i}+\mathbf{j}}{2} + \frac{d}{g}\frac{1+\mathbf{k}}{2} \quad \text{with } g = \mathsf{GCD}(a, b, c, d)$$

Algorithm 4.42 Mul

Input: α: QuatElem
β: QuatElem
Output: $\alpha\beta$: QuatElem
1: $a, b, c, d, e \leftarrow \mathsf{CoordinatesB}(\alpha)$
2: $A, B, C, D, E \leftarrow \mathsf{CoordinatesB}(\beta)$
3: $v_1 \leftarrow (a+b)(A+B)$
4: $t \leftarrow (a-b)(A-B)$
5: $v_2 \leftarrow (v_1 - t)/2$
6: $v_1 \leftarrow -(v_1 + t)/2$
7: $s \leftarrow -(c+d)(C+D)$
8: $t \leftarrow -(c-d)(C-D)$
9: $r \leftarrow (s+t)/2$
10: $s \leftarrow (s-t)/2$
11: $v \leftarrow (a+b+c+d)(A+B+C+D)$
12: $u \leftarrow (a-b+c-d)(A-B+C-D)$
13: $t \leftarrow (u+v)/2 + r + v_1 - 2bD$
14: $u \leftarrow (v-u)/2 + s - v_2 - 2Bc$
15: $r \leftarrow v_1 + 2aA + pr$
16: $s \leftarrow v_2 + p(s + 2cD)$
17: $v \leftarrow eE$
18: **return** $(r + \mathbf{i}s + \mathbf{j}t + \mathbf{k}u)/v$

4.3.1.3. Special order $\mathcal{O}_0$, resolution of norm equations. The RepresentInteger algorithm aims at solving equations of the form $\mathsf{Norm}(\alpha) = M$ in $\mathcal{O}_0$. It builds upon the Cornacchia algorithm introduced later in Section 4.3.2. RepresentInteger relies on Cornacchia returning its solution ordered as $0 \leq x \leq y$, which fixes the pair (x, y) uniquely.

Note that RepresentInteger may fail, and the failure probability depends on the value M/p. For all instances in SQIsign, the failure probability is negligible under the plausible heuristic that the values $M - p(z^2 + t^2)$ behave like random integers of the same size. We give more details on that in Section 9.3.

4.3.2. Gaussian integers

The canonical embedding of the ring of Gaussian integers through the natural inclusion $\mathbb{Z}[\mathbf{i}] \subset B_{p,\infty}$ will be of particular relevance, and we describe below several useful algorithms to handle some operations.

Definition 4.3.2. The type GaussElem represents elements in the subset $\mathbb{Z}[\mathbf{i}] \subset B_{p,\infty}$ of Gaussian integers.

Gaussian Division and GCD.

Algorithm 4.47 GaussDiv(GaussElem, GaussElem): GaussElem

$$(a + \mathbf{i}b, c + \mathbf{i}d) \mapsto \left\lfloor \frac{ac+bd}{c^2+d^2} \right\rceil + \mathbf{i}\left\lfloor \frac{bc-ad}{c^2+d^2} \right\rceil, (a+\mathbf{i}b) - (c+\mathbf{i}d)\left(\left\lfloor \frac{ac+bd}{c^2+d^2} \right\rceil + \mathbf{i}\left\lfloor \frac{bc-ad}{c^2+d^2} \right\rceil\right)$$

4.3.2.1. Cornacchia. Cornacchia's algorithm [Cor08] allows us to efficiently solve norm equations in $\mathbb{Z}[\mathbf{i}]$, or equivalently to find integer solutions to equations of the form $x^2 + y^2 = m$ with m positive integers, provided we know the factorisation of m. For prime m, a pseudocode based on the algorithm SumOfSquares introduced in the next section is given by Cornacchia.

Algorithm 4.46 RepresentInteger(M, prng)

Input: M: Int, odd and such that $M > p$
prng: PRNG
Output: γ: QuatElem, $\gamma \in \mathcal{O}_0$ with $\text{nrd}(\gamma) = M$

```
counter ← 0
bound ← ⌈M/p⌉
repeat
    counter ← counter + 1
    m ← ⌊√(M/p − 1)⌋
    z ← SampleFromInterval(1, m, prng)
    m' ← ⌊√((M − pz²)/p)⌋
    t ← SampleFromInterval(1, m', prng)
    M' ← M − p(z² + t²)
    if IsPrime(M') then
        res ← Cornacchia(M')
        if res ≠ ⊥ then
            x, y ← res                    // 0 ≤ x ≤ y, as guaranteed by Cornacchia
            g ← GCD(x, y, z, t)
            if g == 1 then
                γ ← x + yi + zj + tk
                return γ
until counter ≥ bound
raise Exception ("RepresentInteger failed: no suitable input to Cornacchia found")   // see Section 9.3
```

Algorithm 4.48 GaussGCD

Input: α: GaussElem,
β: GaussElem
Output: μ: GaussElem, $\mu = \text{GCD}_{\mathbb{Z}[i]}(\alpha, \beta)$

```
κ, ρ ← α, β
if nrd(κ) < nrd(ρ) then
    κ, ρ ← ρ, κ
if nrd(ρ) == 1 then
    return ρ
if ρ == 0 then
    return κ
κ, ρ ← GaussDiv(κ, ρ)
return GaussGCD(κ, ρ)
```

Algorithm 4.49 Cornacchia(m)

Input: m: Int, prime
Output: $\perp$ if no solutions to $x^2 + y^2 = m$ exists in $\mathbb{Z}$,
x, y: Int, solution to $x^2 + y^2 = m$ with $0 \le x \le y$ otherwise

```
if −1 is not a square modulo m then
    return ⊥
if m == 2 then
    return 1, 1
r ← ModularSQRT(−1 mod m, m)
x, y ← SumOfSquares(r, m, BitLenCornacchiaInput)
if x > y then
    x, y ← y, x                           // Order the solution as 0 ≤ x ≤ y
return x, y
```

4.3.3. Lattice reduction

This section introduces several useful building blocks that we will apply to the reduction of various types of lattices, based on [ESWvW26]. While our reference implementation aims to be constant-time (using input-independent control flow and fixed-point arithmetic, among other techniques), the corresponding pseudo-code is written in a simpler way for the sake of readability and clarity. Most of our lattice reduction algorithms produce output in a canonical form (with high probability). This is primarily intended to make our algorithms deterministic and parts of the scheme canonical up to, for example, ideal equivalence. We also often require a public bound β_λ on the size of the largest possible input, at a given security level. This is used to determine the algorithm parameters, the size of the intermediate integers, and the fixed-precision parameters that have to be used. Which quantity it bounds, and how large it is, depends on the algorithm: the rank-4 reduction of a MulIdeal bounds the doubled ideal norm $2N$ uniformly by BitLenCommitSkChall $= 2\log_2 p + 4\log_2$ EIBox, whereas the $\mathbb{Z}[\mathbf{i}]$ reduction of an inert ideal bounds the entries of the ideal at hand and therefore differs between call sites. These bounds are required to be large enough that they (heuristically) only fail with probability less than $2^{-\lambda}$. Their concrete instantiations for the protocol are gathered in Table 10.

In this document we use the convention that $\lambda_i(\mathcal{L})$ is the *squared* i-th minimum of the lattice $\mathcal{L}$. A basis $b_1, \ldots, b_n$ of a lattice $\mathcal{L}$ is Minkowski-reduced if for all $1 \leq i \leq n$, b_i is of minimal norm among all the lattice vectors such that $b_1, \ldots, b_i$ can be extended to a basis of $\mathcal{L}$. A basis reaching the successive minima of a lattice is Minkowski-reduced. In dimension $n \leq 4$, a basis reaching the successive minima always exists. We say that a basis of a lattice $\mathcal{L}$ is ϵ-Minkowski-reduced if its vectors b_i satisfy $\|b_i\|^2 \leq (1+\epsilon)\lambda_i(\mathcal{L})$ for all i. Heuristically, an ϵ-Minkowski-reduced basis in the context of this document is Minkowski-reduced, see Section 9.1 for more details.

TABLE 7. List of global parameters for lattice reduction, with their values and appearances

Name	Value	Algorithms
`Z2_thresh`	30	PartialEEAZ LehmerEEAZ LehmerSizeReduction LatReduction2DZ SumOfSquares
`Z2_window`	$63 = 2 \cdot$ `Z2_thresh` $+ 3$	LehmerEEAZ LehmerSizeReduction
`Z2_thresh_gram`	29	GramPartialReduce2D GramLehmerReduce2D DualLatReduction4DZ
`Z2_window_gram`	$61 = 2 \cdot$ `Z2_thresh_gram` $+ 3$	GramPartialReduce2D GramLehmerReduce2D
`Zi_thresh`	27	InnerGLZi LehmerGLZi InertIdealReduction
`Zi_window`	$59 = 2 \cdot$ `Zi_thresh` $+ 5$	InnerGLZi LehmerGLZi
`Up_Shift`	3	InnerGLZi

4.3.3.1. Algorithms for $\mathbb{Z}$-lattices of rank 2. We first target dimension 2 lattices over the integers. Our goal here is to introduce two algorithms to reduce bases of $\mathbb{Z}$-lattices of rank 2. We distinguish lattices given by a basis in HNF shape, reduced with LatReduction2DZ from lattices given by an integral Gram matrix and reduced with GramLehmerReduce2D. The former will be used much later in the ideal-to-isogeny translation, while the latter arise in the reduction of $\mathbb{Z}$-lattices of rank 4.

PartialEEAZ can generally be used with different input types, although we do not guarantee its output then. In a soon-to-be-published note [ESWvW26], we prove that for $\mathrm{bits}(a) = \texttt{Z2\_window}$ and $\mathrm{bits}(b) \geq 64 - \texttt{Z2\_thresh}$, as long as the global threshold H does not stop the loop earlier, $\mathrm{bits}(a') + \mathrm{bits}(b') \leq \mathrm{bits}(a) + \mathrm{bits}(b) - (\texttt{Z2\_thresh} - 1)$ and the t_{ij}'s all satisfy $|t_{ij}| \leq 2^{\texttt{Z2\_thresh}-1}$.

Algorithm 4.50 PartialEEAZ

Input: a, b: Int_{64}, positive, $a \geq b$.
H: Int_{64}, positive, global threshold.
`eea`: Boolean, toggle between EEA or long division
Output: a', b': Int_{64}, positive
$t_{00}, t_{01}, t_{10}, t_{11}$: Int_{32}, where $T = [(t_{ij})]_{0 \leq i \leq 1, 0 \leq j \leq 1}$ satisfies $T(a, b)^t = (a', b')$

1: $t_{00}, t_{01}, t_{10}, t_{11} \leftarrow 1, 0, 0, 1$
2: $\tau \leftarrow \max(H, 2^{64-\texttt{Z2\_thresh}})$ // Usually $H \sim \sqrt{a}$.
3: **for** i **from** 0 **up to** $\texttt{Z2\_thresh} + 1$ **do**
4: **if** $a < b$ **and** `eea` **then** // Do not swap if long division
5: Swap (a, b), swap (t_{00}, t_{10}) and swap (t_{01}, t_{11})
6: **if** $a \geq b$ **and** $b \geq \tau$ **then** // stop if b is too small
7: $k \leftarrow \lfloor \log_2(a/b) \rfloor$
8: $a \leftarrow a - \mathsf{ShiftLeft}(b, k)$
9: $t_{00}, t_{01} \leftarrow t_{00} - \mathsf{ShiftLeft}(t_{10}, k), t_{01} - \mathsf{ShiftLeft}(t_{11}, k)$
10: **return** $a, b, t_{00}, t_{01}, t_{10}, t_{11}$

Lehmer's approach for the Euclidean algorithm is to progressively reduce A, B by using only the most significant words a, b, and applying the resulting (small) transformation matrix on A, B. A caveat is that B may be significantly smaller than A. In this case we instead do the long division of a by b and apply the transformation on A, B, with a suitable realignment to account for the large difference. In any case, we can prove that an iteration reduces the (large) A, B by about $\texttt{Z2\_thresh} - 1$ bits — the details will be available in a public note soon [ESWvW26]. At steps $21, 22$, we are in the long division case. Since $\tau \geq 2^{64-\texttt{Z2\_thresh}}$, the local stopping condition of PartialEEAZ keeps $|t_{ij}| \leq 2^{\texttt{Z2\_thresh}-1}$, and so we apply the shift on the large entries instead. We also use LehmerEEAZ to compute sums of two squares for suitable prime inputs.

For two vectors u, v, we say v is size-reduced with respect to u if $|\langle v, u \rangle| / \|u\|^2 \leq \frac{1}{2}$. A two-dimensional basis (u, v) is size-reduced if v is size-reduced wrt. u. Lehmer's approach also applies to size-reduction, as presented in SizeReductionMSB and LehmerSizeReduction.

We are now ready to introduce the first of our two main lattice reduction algorithms LatReduction2DZ which assumes that the input is given as a HNF basis. Recall that $\lambda_i(\mathcal{L})$ is the i-th successive *squared* minimum of $\mathcal{L}$, taken with respect to the underlying quadratic form. Our algorithm outputs an ϵ-Minkowski-reduced basis, for an ϵ essentially driven by the precision at which the computations are performed. Failures to be exactly Minkowski-reduced are of two kinds: the first vector is exactly shortest unless the minima are almost equal and cannot be distinguished because of limited precision, and the second reaches λ_2 exactly unless the ratio λ_2/λ_1 is too large, in which case our approach fails to fully size-reduce the second vector. The latter situation is unlikely to happen for the lattices[1] considered in this work, and "almost shortest vectors" are enough for the rest of the protocol.

[1] See Heuristic 9.1.1, which for such lattices bounds the probability that $\lambda_1 \leq 2^{-t} \det$ by 2^{-t}; the experimental support will be detailed in a later note.

Algorithm 4.51 LehmerEEAZ

Input: A, B, C: Int, positive, $A \geq B$, so that $\mathbb{Z}(A,0) + \mathbb{Z}(B,C)$ is a HNF-basis of a lattice $\mathcal{L}$.
M: Int, positive, numbers of iterations
Output: A', B', C', D': Int, A', B' positive, $\mathbb{Z}(A',C') + \mathbb{Z}(B',D')$ is another basis with shorter vectors.

$T_{00}, T_{01}, T_{10}, T_{11} \leftarrow 1, 0, 0, 1$
$V \leftarrow AC$
$s_k \leftarrow \lfloor (\mathrm{bitlen}(V) - \texttt{Z2\_thresh})/2 \rfloor$
$\alpha \leftarrow \mathsf{ShiftRight}(\lfloor \sqrt{V} \rfloor, s_k)$ // $\alpha \cdot 2^{s_k} \sim \sqrt{\det \mathcal{L}}$
for i **from** 0 **up to** M **do**
  if $A < B$ **then**
    Swap (A, B), swap (T_{00}, T_{10}) and swap (T_{01}, T_{11})
  $m_A, m_B \leftarrow \mathrm{bitlen}(A), \mathrm{bitlen}(B)$ // Start: select MSBs depending on numbers' size difference
  $\rho \leftarrow m_A - m_B$
  $\texttt{eea} \leftarrow (\rho < \texttt{Z2\_thresh})$ // long division if B is significantly smaller than A
  $s_A \leftarrow \max(m_A - \texttt{Z2\_window}, 0)$
  if eea **then** // adjust shift for long division
    $s_B \leftarrow s_A$
  else
    $s_B \leftarrow m_B - (\texttt{Z2\_window} - \texttt{Z2\_thresh} + 1)$
  $s \leftarrow \max(s_A - s_B, 0)$
  $a, b \leftarrow \mathsf{ShiftRight}(A, s_A), \mathsf{ShiftRight}(B, s_B)$ // Reduce MSBs
  $\hat{\alpha} \leftarrow \lfloor \alpha \cdot 2^{s_k - s_B} \rfloor$ // make α and b on same scale
  $a, b, t_{00}, t_{01}, t_{10}, t_{11} \leftarrow \mathsf{PartialEEAZ}(a, b, \hat{\alpha}, \texttt{eea})$
  if eea **then** // Apply transform globally
    $A, B \leftarrow t_{00}A + t_{01}B, t_{10}A + t_{11}B$
    $T_{00}, T_{01} \leftarrow t_{00}T_{00} + t_{01}T_{10}, t_{00}T_{01} + t_{01}T_{11}$
    $T_{10}, T_{11} \leftarrow t_{10}T_{00} + t_{11}T_{10}, t_{10}T_{01} + t_{11}T_{11}$
  else // Realign numbers in the long division case
    $A \leftarrow A + t_{01} \cdot \mathsf{ShiftLeft}(B, s)$
    $T_{00}, T_{01} \leftarrow T_{00} + t_{01} \cdot \mathsf{ShiftLeft}(T_{10}, s), T_{01} + t_{01} \cdot \mathsf{ShiftLeft}(T_{11}, s)$
  if $A < 0$ **then**
    Negate A, T_{00} and T_{01}.
  if $B < 0$ **then**
    Negate B, T_{10} and T_{11}.
return $A, B, C \cdot T_{01}, C \cdot T_{11}$

Algorithm 4.52 SumOfSquares

Input: m: Int, prime congruent to $1 \bmod 4$.
r: Int, positive with $0 < r < m$ and $r^2 \equiv -1 \bmod m$.
β_λ: Int, positive, a bound on $\mathrm{bitlen}(m)$, the determinant of the lattice $\mathbb{Z}(m,0) + \mathbb{Z}(r,1)$.
Output: x, y: Int, positive, such that $x^2 + y^2 = m$

$M \leftarrow \lfloor \beta_\lambda / (\texttt{Z2\_thresh} - 1) \rfloor + 1$
$A, B, C, D \leftarrow \mathsf{LehmerEEAZ}(m, r, 1, M)$
return $|C|, |D|$

Algorithm 4.53 SizeReductionMSB

Input: a, b, c, d: Int_{64}, a, b positive.
Output: k: Int_{32}, such that $(a, c) - k \cdot (b, d)$ is size-reduced wrt. (b, d)

$s \leftarrow 63 - \max(\mathrm{bitlen}(b), \mathrm{bitlen}(|d|))$
$v_0, v_1 \leftarrow (b, d), (a, c)$
$w_0 \leftarrow 2^s v_0$
$P, Q \leftarrow \langle w_0, v_1 \rangle, \langle w_0, v_0 \rangle$
return $\lfloor \frac{P}{Q} \rceil$

Algorithm 4.54 LehmerSizeReduction

Input: A, B, C, D: Int, A, B positive
M: Int, positive, number of iterations
Output: A', B', C', D': Int, A', B' positive, such that $\big((A', C'), (B', D')\big)$ is a size-reduced basis of $\mathbb{Z}(A, C) + \mathbb{Z}(B, D)$

for i **from** 0 **up to** M **do**
  $\beta_0, \beta_1 \leftarrow \mathrm{bitlen}(|(A, C)|_\infty), \mathrm{bitlen}(|(B, D)|_\infty)$
  $\beta \leftarrow \max(\beta_0, \beta_1)$
  $\rho \leftarrow \beta - \beta_1 + 1$
  $s_0 \leftarrow \max(\beta - \texttt{Z2\_window}, 0)$
  $s \leftarrow \max(\rho - \texttt{Z2\_thresh}, 0)$
  if $s > 0$ **then**
    $s_1 \leftarrow \beta_1 - (\texttt{Z2\_window} - \texttt{Z2\_thresh} + 1)$
  else
    $s_1 \leftarrow s_0$
  $a, b, c, d \leftarrow \mathsf{ShiftRight}(A, s_0), \mathsf{ShiftRight}(B, s_1), \mathsf{ShiftRight}(C, s_0), \mathsf{ShiftRight}(D, s_1)$
  $\mu \leftarrow \mathsf{SizeReductionMSB}(a, b, c, d)$
  $A, C \leftarrow A - \mu \cdot \mathsf{ShiftLeft}(B, s), C - \mu \cdot \mathsf{ShiftLeft}(D, s)$
  If $A < 0$, negate A and C
  If $B < 0$, negate B and D
return A, B, C, D

Algorithm 4.55 LatReduction2DZ

Input: A, B, C: Int, positive, with $B < A$, and $\mathbb{Z}(A, 0) + \mathbb{Z}(B, C)$ is a HNF basis of a lattice
β_λ: Int, positive, a bound on $\mathrm{bitlen}(AC)$, the determinant of the lattice
Output: A', B', C', D': Int, $A' \geq 0, B' \geq 0$ and $\big((A', C'), (B', D')\big)$ is a $2^{3-\texttt{2Z2\_thresh}}$-Minkowski-reduced basis of the same lattice

$M \leftarrow \lfloor \beta_\lambda / (\texttt{Z2\_thresh} - 1) \rfloor + 1$
$A, B, C, D \leftarrow \mathsf{LehmerEEAZ}(A, B, C, M)$
$A, B, C, D \leftarrow \mathsf{LehmerSizeReduction}(A, B, C, D, M)$
$A, B, C, D \leftarrow \mathsf{LehmerSizeReduction}(B, D, A, C, 1)$
if $A^2 + C^2 > B^2 + D^2$ **then**
  return B, A, D, C
else
  return A, B, C, D

For the GramLehmerReduce2D algorithm (that takes the Gram matrix of the lattice as input), we continue our usage of Lehmer's variant, relying on a partial Gram reduction GramPartialReduce2D, very close in spirit to PartialEEAZ.

Algorithm 4.56 GramPartialReduce2D

Input: a, b, c: Int_{64}, a, c positive, entries of a matrix $G = [(a, b), (b, c)]$.
`eea`: Boolean, allowing swaps or not.
Output: a', b', c': Int_{64}, positive.
$t_{00}, t_{01}, t_{10}, t_{11}$: Int_{32}, with $T = [(t_{ij})]_{0 \le i \le 1, 0 \le j \le 1}$ such that $TGT^t = [(a', b'), (b', c')]$ and $\det T = \pm 1$.

$t_{00}, t_{01}, t_{10}, t_{11} \leftarrow 1, 0, 0, 1$
$N \leftarrow \lfloor \texttt{Z2\_thresh\_gram}/2 \rfloor + 2$
$\tau \leftarrow 2^{\texttt{Z2\_window\_gram} - \texttt{Z2\_thresh\_gram}}$
for i **from** 0 **up to** N **do**
  if $b < 0$ **then**
    Negate b, t_{00}, t_{01}
  if $c < a$ **and** `eea` **then**
    Swap (a, c), swap (t_{00}, t_{10}) and swap (t_{01}, t_{11})
  if $2b > a$ **and** $a \ge \tau$ **then** // stop if the basis is reduced or a too small
    $k \leftarrow \max(\lfloor \log_2(b/a) \rfloor, 0)$
    $b, c \leftarrow b - \mathsf{ShiftLeft}(a, k), c - \mathsf{ShiftLeft}(b, k+1) + \mathsf{ShiftLeft}(a, 2k)$ // Update: $G \leftarrow TGT^t$
    $t_{10}, t_{11} \leftarrow t_{10} - \mathsf{ShiftLeft}(t_{00}, k), t_{11} - \mathsf{ShiftLeft}(t_{01}, k)$
return $a, b, c, t_{00}, t_{01}, t_{10}, t_{11}$

Algorithm 4.57 GramLehmerReduce2D

Input: A, B, C: Int, A, C positive, entries of a postive definite matrix $G = [(A, B), (B, C)]$
M: Int, positive, number of iterations
Output: A', B', C': Int, A', C' positive.
$T_{00}, T_{01}, T_{10}, T_{11}$: Int, with $T = [(T_{ij})]$ such that $TGT^t = [(A', B'), (B', C')]$ and $\det T = \pm 1$.

$T_{00}, T_{01}, T_{10}, T_{11} \leftarrow 1, 0, 0, 1$
for i **from** 0 **up to** M **do**
  if $C < A$ **then**
    Swap (A, C), (T_{00}, T_{10}) and (T_{01}, T_{11})
  $\beta_A, \beta_C \leftarrow \mathrm{bitlen}(A), \mathrm{bitlen}(C)$ // Select MSBs depending on numbers' size difference
  $\rho \leftarrow \beta_C - \beta_A$
  `eea` $\leftarrow (\rho <$ `Z2_thresh_gram`$)$
  $s_C \leftarrow \max(\beta_C -$ `Z2_window_gram`$, 0)$
  $s \leftarrow \max(\lfloor (\rho -$ `Z2_thresh_gram` $+ 2)/2 \rfloor, 0)$
  $s_A, s_B \leftarrow s_C - 2s, s_C - s$
  $a, b, c \leftarrow \mathsf{ShiftRight}(A, s_A), \mathsf{ShiftRight}(B, s_B), \mathsf{ShiftRight}(C, s_C)$ // Reduce the Gram defined by the MSBs
  $a, b, c, t_{00}, t_{01}, t_{10}, t_{11} \leftarrow \mathsf{GramPartialReduce2D}(a, b, c,$ `eea`$)$
  if `eea` **then** // Apply transform globally
    $U \leftarrow [t_{ij}]$
    $G \leftarrow UGU^t$
    $A, B, C \leftarrow G_{00}, G_{01}, G_{11}$
    $T \leftarrow UT$ // $T := [T_{ij}]$
  else
    $B, C \leftarrow B + t_{10} \cdot \mathsf{ShiftLeft}(A, s), C + t_{10} \cdot \mathsf{ShiftLeft}(B, s+1) + t_{10}^2 \cdot \mathsf{ShiftLeft}(A, 2s)$
    $T_{10}, T_{11} \leftarrow t_{10} \cdot \mathsf{ShiftLeft}(T_{00}, s) + t_{11} T_{10}, t_{10} \cdot \mathsf{ShiftLeft}(T_{01}, s) + t_{11} T_{11}$
return $A, B, C, T_{00}, T_{01}, T_{10}, T_{11}$

4.3.3.2. Algorithms for $\mathbb{Z}[\mathbf{i}]$-lattices (of rank 2). Due to the presence of $\mathbf{i}$ in the maximal order $\mathcal{O}_0$, the left $\mathcal{O}_0$-ideals can be interpreted as $\mathbb{Z}[\mathbf{i}]$-lattices of rank 2. Using that, we can derive more efficient reduction algorithms. Our main algorithm for reducing these lattices is LehmerGLZi.

For $u \in \mathbb{C}^d$, write $|u|_\infty$ for the ℓ_∞-norm of u seen in $\mathbb{R}^{2d}$. For $u, v \in \mathbb{Z}[\mathbf{i}]^2$, write $\langle u, v\rangle = \overline{u_1}v_1 + \overline{u_2}v_2$ for the Hermitian inner product. The next algorithm LehmerGLZi always makes sure such inputs are passed to InnerGLZi. The output is then an ϵ-Minkowski-reduced basis (for the minima as a $\mathbb{Z}[\mathbf{i}]$-lattice). Failures to be exactly Minkowski-reduced are identical[2] to those in the previous section, with the second vector reaching λ_2 exactly as soon as $\lambda_2/\lambda_1 \leq 2^{2(\texttt{Zi\_thresh}-1)}$.

Algorithm 4.58 InnerGLZi

Input: a, b, c, d: GaussElem$_{64}$.
`eea`: Boolean, allowing swaps or not.
Output: a', b', c', d': GaussElem
$t_{00}, t_{01}, t_{10}, t_{11}$: GaussElem, where $T = [(t_{ij})]_{0\leq i\leq 1, 0\leq j\leq 1}$ satisfies $T(a, b)^t = (a', b')^t$ and $T(c, d)^t = (c', d')^t$

$v_0, v_1 \leftarrow (a, c), (b, d)$
$t_{00}, t_{01}, t_{10}, t_{11} \leftarrow 1, 0, 0, 1$
$q \leftarrow 1$
$\tau \leftarrow 64 + \texttt{Zi\_thresh} - \texttt{Zi\_window} - \texttt{Up\_Shift}$
for i **from** 0 **up to** $\texttt{Zi\_thresh} + 1$ **do**
  $s_0 \leftarrow 64 - \texttt{Up\_Shift} - \text{bitlen}(|(a, c)|_\infty)$
  $s_1 \leftarrow 64 - \texttt{Up\_Shift} - \text{bitlen}(|(b, d)|_\infty)$
  if $\max(s_0, s_1) \leq \tau$ **and** $q > 0$ **then** // Do nothing if vectors are too short or we have finished reduction
    $w_0, w_1 \leftarrow 2^{s_0}v_0, 2^{s_1}v_1$
    $\delta \leftarrow s_0 - s_1$
    if $\|w_0\| > 2^\delta \|w_1\|$ **and** `eea` **then** // Maintain first vector as the shorter one
      Swap (v_0, v_1), (w_0, w_1), swap (t_{00}, t_{10}) and swap (t_{01}, t_{11})
      $\delta \leftarrow -\delta$
    $P, Q \leftarrow \langle w_0, w_1\rangle, \|w_0\|^2$
    $\eta \leftarrow \lceil \log_2\left(|P|_\infty\right) - \log_2 Q\rceil - 1$
    $k \leftarrow \max(\delta + \eta, 0)$
    $\sigma \leftarrow \delta - k$
    $q \leftarrow \lfloor 2^\sigma \cdot \frac{P}{Q}\rceil$ // $2^\sigma \frac{P}{Q} \in [-2, 2] + \mathbf{i}[-2, 2]$
    $v_1 \leftarrow v_1 - q \cdot (2^k v_0)$
    $t_{10}, t_{11} \leftarrow t_{10} - q \cdot (2^k t_{00}), t_{11} - q \cdot (2^k t_{01})$
Parse (a, c) from v_0, (b, d) from v_1
return $a, b, c, d, t_{00}, t_{01}, t_{10}, t_{11}$

[2] A distinction is the $\mathbb{Z}[\mathbf{i}]$ tail bound of Heuristic 9.1.1, under which the probability that $\lambda_1 \leq 2^{-2t}$ det is less than 2^{-4t}; the experimental support will be detailed in a later note.

Algorithm 4.59 LehmerGLZi

Input: A, B, C, D: GaussElem.
M: Int, positive, number of iterations.
Output: A', B', C', D': GaussElem, with $\Re A', \Im A' \geq 0$ and $\Re H, \Im H \geq 0$ for $H = \langle (A', C'), (B', D') \rangle$;
$(A', C'), (B', D')$ is a $2^{5-2\texttt{Zi\_thresh}}$-Minkowski-reduced basis of $\mathbb{Z}[\mathbf{i}](A, C) + \mathbb{Z}[\mathbf{i}](B, D)$.

for $0 \leq i < M$ **do**
  $\beta_0, \beta_1 \leftarrow \text{bitlen}(|(A, C)|_\infty), \text{bitlen}(|(B, D)|_\infty)$
  If $\beta_0 > \beta_1$, swap (A, C) and (B, D) // Maintain smaller vector as the first basis vector
  $\rho \leftarrow \beta_1 - \beta_0 + 1$
  `eea` $\leftarrow (\rho <$ `Zi_thresh`$)$
  $s_1 \leftarrow \max(\beta_1 -$ `Zi_window`$, 0)$
  if `eea` **then**
    $s_0 \leftarrow s_1$
  else
    $s_0 \leftarrow \max(s_1 -$ (`Zi_window` $-$ `Zi_thresh` $+ 1), 0)$
  $s \leftarrow \rho -$ `Zi_thresh` // variable for extra shift in case of large difference in size
  $a, c \leftarrow \text{ShiftRight}(A, s_0), \text{ShiftRight}(C, s_0)$
  $b, d \leftarrow \text{ShiftRight}(B, s_1), \text{ShiftRight}(D, s_1)$
  $a, b, c, d, t_{00}, t_{01}, t_{10}, t_{11} \leftarrow$ InnerGLZi$(a, b, c, d,$ `eea`$)$
  if `eea` **then**
    $A, B \leftarrow t_{00}A + t_{01}B, t_{10}A + t_{11}B$
    $C, D \leftarrow t_{00}C + t_{01}D, t_{10}C + t_{11}D$
  else // Realign numbers in the long division case
    $B, D \leftarrow B + t_{01} \cdot \text{ShiftLeft}(A, s), D + t_{01} \cdot \text{ShiftLeft}(C, s)$
Find $u \in \{\pm 1, \pm\mathbf{i}\}$ so that $\Re uA, \Im uA \geq 0$, $(A, C) \leftarrow u \cdot (A, C)$
Find $u \in \{\pm 1, \pm\mathbf{i}\}$ so that $\Re uH, \Im uH \geq 0$ for $H = \langle (A, C), (B, D) \rangle$, $(B, D) \leftarrow u \cdot (B, D)$
return A, B, C, D

4.3.3.3. Algorithm for $\mathbb{Z}$-lattices of rank 4. The reduction of four-dimensional generic $\mathbb{Z}$-lattices is needed to sample small elements in generic quaternion ideals, corresponding to objects of the MulIdeal type defined in the next section. For that, our main reduction algorithm is DualLatReduction4DZ which we present next. This algorithm builds upon several building blocks that are introduced afterward. DualLatReduction4DZ will be used in the next section in MulIdealDualReduction.

Efficiency is achieved by reducing the *dual* of the corresponding lattice and keeping the inverse transformation T describing the reduction. As in [HKK+25], our reduction strategy in DualLatReduction4DZ uses a variant of BKZ-2 on the Gram matrix with a fixed number of iterations, followed by a Minkowski reduction (on an already well-reduced basis). Our approach uses the LDL decomposition: a Gram matrix G can be written uniquely as $G = LDL^t$ with L lower triangular with 1 on the diagonal and D diagonal positive. We avoid the costly computation of the dual Gram matrix G^{-1} thanks to an implicit scaling (by the known $\det G$) and the use of the Gram matrix $G^\vee$ of the *reversed* dual basis instead. The LDL decomposition of $G^\vee$ can be obtained very efficiently from the HNF-basis of a MulIdeal. The initial profile $((2N)^2, (2N)^2, p, p)$ is expressed in squared lengths, so its entries span at most $\gamma = 2\beta_\lambda - \text{bitlen}(p)$ bits, and this γ plays for each block reduction the role that the determinant bound plays in LatReduction2DZ: an iteration of GramLehmerReduce2D removes about `Z2_thresh_gram` bits of the gap between the two diagonal entries of its block, whence the choice of M below. The number K of tours, on the other hand, is heuristic; see Heuristic 9.1.2. The underlying lattice bases are size-reduced if and only if the non-diagonal entries of L have magnitude at most $\frac{1}{2}$, and we also say that L is size-reduced in this case.

A lattice basis is BKZ_2-reduced if the i-th Gram-Schmidt vector b_i^* is the shortest vector of the local projection of $\mathbb{Z}\, b_i + \mathbb{Z}\, b_{i+1}$, for $0 \leq i \leq 2$ — this implies that the basis is $\text{LLL}_{\frac{3}{4}}$-reduced. In dimension 4, one can then show that the vectors of a Minkowski-reduced basis must have small coordinates in a BKZ_2-reduced basis. BKZ2-EvenOdd outputs (a representation of) a BKZ_2-reduced basis of the lattice spanned by its input, except with probability $2^{-\lambda}$. MinkowskiReduce exploits the above fact with a greedy exploration of candidates, similar to [NS09]; if its input is BKZ_2-reduced, we can prove that its output is Minkowski-reduced, except in one extreme but identified case. Indeed, if the input is a basis of a lattice similar[3] to the lattice D_4, MinkowskiReduce could output a transformation only

[3] A similarity is the composition of a scaling and an isometry.

Algorithm 4.60 DualLatReduction4DZ

Input: N, A, B, C, E: Int, $0 \le A, B, C, E < 2N$ and $2N\,\mathbb{Z} + 2N\mathbf{i}\,\mathbb{Z} + (A + B\mathbf{i} + \mathbf{j})\,\mathbb{Z} + (C + E\mathbf{i} + \mathbf{j})\,\mathbb{Z}$ is a HNF-basis H of $2I$, where I is a MulIdeal of $B_{p,\infty}$
β_λ: Int, positive, a bound on $\mathrm{bitlen}(2N)$

Output: $\lambda_1, \lambda_2, \lambda_3, \lambda_4$: Int, positive, the squared successive minima of $(4N^2p) \cdot H^\vee$, except with probability $2^{-\lambda}$, where $H^\vee$ is the reversed dual basis of H
$T \in \mathbb{Z}^{4\times4}$, with $\det T = \pm1$, such that $T^{-1}H^\vee$ is, except with probability $2^{-\lambda}$, a Minkowski-reduced basis of the dual, canonical up to a global sign

1: $\ell \leftarrow -\frac{1}{2N} \cdot \begin{bmatrix} E & B \\ C & A \end{bmatrix}$
2: $L, D \leftarrow \begin{bmatrix} I_2 & 0 \\ \ell & I_2 \end{bmatrix}, \mathrm{diag}((2N)^2, (2N)^2, p, p)$ // $LDL^t = (2N)^2p \cdot G^\vee$
3: $\gamma \leftarrow 2\beta_\lambda - \mathrm{bitlen}(p)$ // Bound on the gap of the diagonal profile
4: $K, M \leftarrow \lfloor \log_2 \gamma \rfloor + 4, \lfloor \gamma/(\texttt{Z2\_thresh\_gram} - 1) \rfloor + 1$ // Tours, and iterations per block reduction
5: $L, D, T \leftarrow$ BKZ2-EvenOdd(L, D, K, M)
6: $\lambda_1, \lambda_2, \lambda_3, \lambda_4, T \leftarrow$ MinkowskiReduce(L, D, T)
7: **return** $(\lambda_1, \lambda_2, \lambda_3, \lambda_4, T)$

Algorithm 4.61 BKZ2-EvenOdd

Input: $L, D \in \mathbb{Q}^{4\times4}$, L lower triangular with 1 on the diagonal, D diagonal positive
K: Int, positive, number of iterations
M: Int, positive, number of iterations per block

Output: $L', D' \in \mathbb{Q}^{4\times4}$, L' lower triangular with 1 on the diagonal, D' diagonal positive, such that $L'\sqrt{D'}$ is a $\mathtt{BKZ}_2$-reduced basis of the lattice spanned by $L\sqrt{D}$, except with probability $2^{-\lambda}$
$T \in \mathbb{Z}^{4\times4}$, with $T \cdot (L'D'L'^t) \cdot T^t = LDL^t$ and $\det T = \pm1$

1: $T \leftarrow I_4$
2: **for** $0 \le i < K$ **do**
3: $\quad L, D, T \leftarrow$ ReduceLocalBlock$(L, D, T, M, 1)$
4: $\quad L, D, T \leftarrow$ ReduceLocalBlock$(L, D, T, M, 0)$
5: $\quad L, D, T \leftarrow$ ReduceLocalBlock$(L, D, T, M, 2)$
6: **return** L, D, T

Algorithm 4.62 ReduceLocalBlock

Input: $L, D \in \mathbb{Q}^{4\times4}$, L lower triangular with 1 on the diagonal, D diagonal positive
$T \in \mathbb{Z}^{4\times4}$, with $\det T = \pm1$
M: Int, positive, a number of iterations
i: Int, with $0 \le i \le 2$, a block index

Output: $L', D' \in \mathbb{Q}^{4\times4}$, L' lower triangular with 1 on the diagonal and size-reduced, D' diagonal positive
$T' \in \mathbb{Z}^{4\times4}$, with $T' \cdot (L'D'L'^t) \cdot T'^t = LDL^t$ and $\det T' = \pm1$

1: $A, B, C \leftarrow D_i, L_{i+1,i}D_i, L_{i+1,i}^2 D_i + D_{i+1}$
2: $A, B, C, t_{00}, t_{01}, t_{10}, t_{11} \leftarrow$ GramLehmerReduce2D(A, B, C, M)
3: $L, D, T \leftarrow$ UpdateLDL$(L, D, T, i, t_{00}, t_{01}, t_{10}, t_{11})$
4: $L, T \leftarrow$ SizeReduction4DZ(L, T, i)
5: **return** L, D, T

to a sublattice. Such inputs cannot occur for the quaternion lattices that we are handling, thus MinkowskiReduce ends up with an (implicit) Minkowski-reduced basis of its input lattice, except with probability $2^{-\lambda}$. The sign rule MinkowskiReduce applies makes the output canonical over the ideal class up to one global sign, which no such rule can fix.

Algorithm 4.63 UpdateLDL

Input: $L, D \in \mathbb{Q}^{4\times 4}$, L lower triangular with 1 on the diagonal, D diagonal positive
$T \in \mathbb{Z}^{4\times 4}$, with $\det T = \pm 1$
i: Int, with $0 \le i \le 2$, a block index
$t_{00}, t_{01}, t_{10}, t_{11}$: Int, such that $t_{00}t_{11} - t_{10}t_{01} = \pm 1$
Output: $L', D' \in \mathbb{Q}^{4\times 4}$, L' lower triangular with 1 on the diagonal, D' diagonal positive
$T' \in \mathbb{Z}^{4\times 4}$, with $T' \cdot (L'D'L'^t) \cdot T'^t = LDL^t$ and $\det T' = \pm 1$

$\mu \leftarrow L_{i+1,i}$ // Recompute local Gram
$u_0, u_1 \leftarrow t_{00} + \mu t_{01}, t_{10} + \mu t_{11}$
$g_0 \leftarrow D_i u_0^2 + D_{i+1} t_{01}^2$
$g_{01} \leftarrow D_i u_0 u_1 + D_{i+1} t_{01} t_{11}$
$L_{i+1,i} \leftarrow \frac{g_{01}}{g_0}$ // Recompute local LDL
$D_i, D_{i+1} \leftarrow g_0, \frac{D_i D_{i+1}}{g_0}$
$\ell_{\text{old}}, \ell_{\text{new}} \leftarrow \begin{bmatrix} 1 & 0 \\ -\mu & 1 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ L_{i+1,i} & 1 \end{bmatrix}$
$d, U \leftarrow t_{00}t_{11} - t_{10}t_{01}, \begin{bmatrix} t_{11} & -t_{10} \\ -t_{01} & t_{00} \end{bmatrix}$ // $dU = [t_{ij}]^{-1}$
for $0 \le j < i$ **do** // Finish update of L
  $L_{i,j}, L_{i+1,j} \leftarrow t_{00}L_{i,j} + t_{01}L_{i+1,j}, t_{10}L_{i,j} + t_{11}L_{i+1,j}$
for $i + 2 \le j < 4$ **do**
  $(L_{j,i}, L_{j,i+1}) \leftarrow d \cdot (L_{j,i}, L_{j,i+1}) \cdot \ell_{\text{old}} \cdot U \cdot \ell_{\text{new}}$
for $0 \le j < 4$ **do** // Update inverse transform
  $(T_{j,i}, T_{j,i+1}) \leftarrow d \cdot (T_{j,i}t_{11} - T_{j,i+1}t_{10}), d \cdot (T_{j,i+1}t_{00} - T_{j,i}t_{01})$
return L, D, T

Algorithm 4.64 SizeReduction4DZ

Input: $L \in \mathbb{Q}^{4\times 4}$, L lower triangular with 1 on the diagonal
$T \in \mathbb{Z}^{4\times 4}$, with $\det T = \pm 1$
i: Int, with $0 \le i \le 2$, a block index
Output: $L' \in \mathbb{Q}^{4\times 4}$ lower triangular with 1 on the diagonal, size-reduced
$T' \in \mathbb{Z}^{4\times 4}$, with $T'L' = L$ and $\det T' = \pm 1$

for $i \le r < 4$ **do**
  for j **from** $r - 1$ **down to** 0 **do**
    $q \leftarrow \lfloor L_{r,j} \rceil$
    for $0 \le k < j$ **do**
      $L_{r,k} \leftarrow L_{r,k} - qL_{j,k}$
    $L_{r,j} \leftarrow L_{r,j} - q$
    for $0 \le k < 4$ **do** // Update the inverse transform
      $T_{k,j} \leftarrow T_{k,j} + qT_{k,r}$
return L, T

Algorithm 4.65 MinkowskiReduce

Input: $L, D \in \mathbb{Q}^{4\times 4}$, L lower triangular with 1 on the diagonal, D diagonal positive such that $L\sqrt{D}$ is $\mathtt{BKZ}_2$-reduced
$T \in \mathbb{Z}^{4\times 4}$, with $\det T = \pm 1$

Output: $\lambda_1, \lambda_2, \lambda_3, \lambda_4$: Int, positive, the squared successive minima of the lattice spanned by $L\sqrt{D}$.
$T' \in \mathbb{Z}^{4\times 4}$, such that $(T')^{-1}L\sqrt{D}$ is Minkowski-reduced, and $\det T' = \pm 1$. Its rows $b_1, \ldots, b_4$ satisfy $b_i b_1^t \geq 0$ for $2 \leq i \leq 4$.

$G, U \leftarrow LDL^t, I_4$
$\mathcal{N} \leftarrow \emptyset$
$\mathcal{S} \leftarrow$ the 13 non-zero triples of $\{\pm 1, 0\}^3$ whose first non-zero entry is positive
for $(c_1, c_2, c_3) \in \mathcal{S}$ **do** // Build a list of candidate short vectors.
    $s \leftarrow \sum_{i=1}^{3} c_i L_{i0}$
    for $c_0 \in \{\lfloor -s \rfloor, \lceil -s \rceil\}$ **do** // Two minimal choices for candidate norm.
        $c \leftarrow (c_0, \ldots, c_3)$
        $\mathcal{N} \leftarrow \mathcal{N} \cup \{(c, cGc^t)\}$
$\mathcal{N} \leftarrow \mathcal{N} \cup \{((1,0,0,0), G_{00})\}$
Sort $\mathcal{N}$ by increasing second component // $\mathcal{N} = \{(u_1, N_1), \ldots, (u_{27}, N_{27})\}$
Set the first row U_1 of U as u_1.
$k \leftarrow 2$
for i **from** 2 **up to** 3 **do** // Greedily explore $\mathcal{N}$ until U leads to a basis.
    repeat
        $u \leftarrow u_k$
        $k \leftarrow k + 1$
        $M \leftarrow [U_1, \ldots, U_{i-1}, u]$ (row matrix)
    until some $i \times i$ minor of M is non-zero. // $U_1, \ldots, U_{i-1}, u$ independent
    Set the i-th row U_i of U as u.
repeat
    $u \leftarrow u_k$
    $k \leftarrow k + 1$
    $M \leftarrow [U_1, U_2, U_3, u]$
until $\det M = \pm 1$
Set the last row of U as u.
For $2 \leq i \leq 4$, negate U_i if $U_i G U_1^t < 0$ // Canonical signs
$T \leftarrow T \cdot U^{-1}$ // U has small entries and $\det U = \pm 1$, inversion is ok.
$\lambda_1, \lambda_2, \lambda_3, \lambda_4 \leftarrow U_1 G U_1^t, \ldots, U_4 G U_4^t$.
return $(\lambda_1, \lambda_2, \lambda_3, \lambda_4, T)$.

4.3.4. Quaternion ideals

This section introduces the supporting algorithms for handling quaternion ideals in SQISIGN. We are going to use several specific representations for quaternion ideals and use tailored algorithms for each of them. First, we introduce a data type which is going to be common to all the representations.

Definition 4.3.3. LatticeBasis consists of four QuatElem values defining a basis of a quaternion ideal in $B_{p,\infty}$.

4.3.4.1. Inert ideals. Following [Ler26] and [LS26], we will make good use of the inert ideal representation, which is compact, suitable for most $\mathcal{O}_0$-ideals and comes with a canonical LatticeBasis representation which is easily computable.

Definition 4.3.4. InertIdeal consists of a triple of integers $I = (N, (x, y))$ with the constraint that $0 \leq x, y < N$, and IdealNorm$(I) = N$.

Algorithm 4.66 InertIdealBasis(InertIdeal) : LatticeBasis

$$(N, (x, y)) \mapsto \left\langle N, N\mathbf{i}, x + y\mathbf{i} + \frac{\mathbf{i}+\mathbf{j}}{2}, -y + x\mathbf{i} + \frac{\mathbf{k}-1}{2} \right\rangle$$

Most importantly, we require a way to compute the inert ideal representation from a generator of the ideal I (an element such that $I = \langle \gamma, N \rangle$ for some $\gamma \in B_{p,\infty}$ and $N \in \mathbb{N}$). Below, we detail several algorithms for doing so for various values of N. An intersection algorithm for inert ideals will also be extremely useful. More explanations and analysis on these algorithms can be found in [LS26].

Algorithm 4.67 InertGenToIdealOdd

Input: N : Int, positive, odd, and coprime to p
γ : QuatElem, with $\mathrm{nrd}(\gamma) = 0 \mod N$ and $\gamma \in \mathcal{O}_0$
Output: $\perp$, if $\mathcal{O}_0\langle \gamma, N \rangle$ is not inert of norm N
I : InertIdeal, with $I = \mathcal{O}_0\langle \gamma, N \rangle$ otherwise

$a, b, c, d, e \leftarrow$ CoordinatesB(ModO$_0(\gamma, N)$)
$s \leftarrow (c)^2 + (d)^2 \mod N$
if GCD$(s, N) \neq 1$ **then**
  $N_2 \leftarrow N/\mathrm{GCD}(s, N)$
  while GCD$(s, N_2) \neq 1$ **do**
    $N_2 \leftarrow N_2/\mathrm{GCD}(s, N_2)$ // at the end of the loop N_2 will be the greatest divisor of N coprime to s
  $\gamma \leftarrow$ Mul $(N_2 + j, e\gamma)$
  $a, b, c, d, e \leftarrow$ CoordinatesB(γ) // e must be 1 here
  $a, b, c, d \leftarrow a, b, c, d \mod N$
  $s \leftarrow c^2 + d^2$
if GCD$(s, N) \neq 1$ **then**
  return $\perp$
$t \leftarrow$ ModularInverse$(2s, N)$
$x \leftarrow (ac + bd)t \mod N$
$y \leftarrow ((bc - ad) - s)t \mod N$
return $(N, (x, y))$

4.3.4.2. Inert ideals reduction. Another useful algorithm will be reduction of inert ideals. The algorithm introduced below for that task is inspired by [ESWvW26], and crucially considers $\mathcal{O}_0$-ideals as $\mathbb{Z}[\mathbf{i}]$-ideals of dimension 2 through the isomorphism between $B_{p,\infty}$ and the two-dimensional $\mathbb{Q}[\mathbf{i}]$ space generated by $(1, \mathbf{j})$.

Let I be an inert ideal described by $(N, (x, y))$, and let $X = 2x, Y = 2y + 1$. From the description above, one checks that $I = \mathbb{Z}[\mathbf{i}]N + \frac{1}{2}\mathbb{Z}[\mathbf{i}](X + Y\mathbf{i} + \mathbf{j})$. In coordinates $1, j$ of $B_{p,\infty}$ as a $\mathbb{Q}(\mathbf{i})$-space, this means $(N, 0), \frac{1}{2}(X + Y\mathbf{i}, 1)$ is a HNF-shaped $\mathbb{Z}[\mathbf{i}]$-basis of I which allows us to use LehmerGLZi.

Algorithm 4.68 InertGenToIdealPowTwo

Input: e: Int, positive
γ: QuatElem, with $\mathrm{nrd}(\gamma) = 0 \mod 2^e$ and $\gamma \in \mathcal{O}_0$
Output: $\perp$, if $\mathcal{O}_0\langle\gamma, 2^e\rangle$ is not inert of norm 2^e
I: InertIdeal, with $I = \mathcal{O}_0\langle\gamma, 2^e\rangle$ otherwise

$t, x, y, z \leftarrow \mathsf{CoordinatesO_0}(\mathsf{ModO_0}(\gamma, 2^e))$
if $y, z \bmod 2 \notin \{(0,1),(1,0)\}$ **then**
  if $y, z \bmod 2 == (1,1)$ **and** $x \neq t \bmod 2$ **then**
    $\gamma \leftarrow \mathsf{Mul}\left(\frac{\mathbf{i}+\mathbf{j}}{2}, \gamma\right)$
    $t, x, y, z \leftarrow \mathsf{CoordinatesO_0}(\mathsf{ModO_0}(\gamma, 2^e))$
  else
    return $\perp$
$s \leftarrow y^2 + z^2 \bmod 2^e$
$s \leftarrow \mathsf{ModularInverse}(s, 2^e)$
$X \leftarrow t(ys + zy + yz) \bmod 2^e$
$Y \leftarrow s(yx - zt - z^2) \bmod 2^e$
return $(2^e, (X, Y))$

Algorithm 4.69 InertIntersect

Input: $I_1 = (N_1, (x_1, y_1))$: InertIdeal
$I_2 = (N_2, (x_2, y_2))$: InertIdeal
Output: I: InertIdeal, with $I_1 \cap I_2 = I$, if $\gcd(N_1, N_2) = 1$
$\perp$, otherwise

if $\mathsf{GCD}(N_1, N_2) \neq 1$ **then**
  return $\perp$
$X \leftarrow \mathsf{CRT}(x_1, N_1, x_2, N_2)$
$Y \leftarrow \mathsf{CRT}(y_1, N_1, y_2, N_2)$
return $(N_1 N_2, (X, Y))$

Algorithm 4.70 InertGenToIdeal

Input: N: Int, positive
γ: QuatElem, with $\mathrm{nrd}(\gamma) = 0 \mod N$ and $\gamma \in \mathcal{O}_0$
Output: $\perp$, if $\mathcal{O}_0\langle\gamma, N\rangle$ is not inert of norm N
I: InertIdeal, with $I = \mathcal{O}_0\langle\gamma, N\rangle$ otherwise

$e \leftarrow \mathsf{DyadicValuation}(N)$
$I_1 \leftarrow \mathsf{InertGenToIdealOdd}(N/2^e, \gamma)$
$I_2 \leftarrow \mathsf{InertGenToIdealPowTwo}(e, \gamma)$
if $I_1 == \perp$ **or** $I_2 == \perp$ **then**
  return $\perp$
return $\mathsf{InertIntersect}(I_1, I_2)$

Algorithm 4.71 PrimeNormIdealInertSampling

Input: N : Int, positive odd prime
prng : PRNG
Output: I : InertIdeal, a non-principal $\mathcal{O}_0$-ideal of norm N sampled with small bias

while `true` **do**
  $x \leftarrow \mathsf{SampleFromInterval}(0, N-1, \mathsf{prng})$
  if $-4x^2 - p$ is a square $\bmod 4N$ **then**
    $t \leftarrow -4x^2 - p \bmod 4N$
    $r \leftarrow \mathsf{ModularSQRT}(t \bmod N, N)$
    $r_4 \leftarrow t \bmod 2$
    $Y \leftarrow \mathsf{CRT}(r_4, 4, r, N)$
    $b \leftarrow \mathsf{SampleFromInterval}(0, 1, \mathsf{prng})$
    $Y \leftarrow (-1)^b Y \mod 2N$
    $y \leftarrow (Y-1)/2$
    return $(N, (x, y))$

Algorithm 4.72 InertIdealReduction

Input: I : InertIdeal, so a triple $(N, (x, y))$ as above
β_λ : Int, positive, a bound on $\max(\mathrm{bitlen}(2N), \mathrm{bitlen}(\widehat{q}))$
Output: B' : LatticeBasis, a Minkowski-reduced basis of I

$\widehat{q} \leftarrow \lfloor \sqrt{p} \rfloor$
$X, Y \leftarrow 2x, 2y+1$
$\rho \leftarrow \lfloor (\max(\beta_\lambda - \mathrm{bitlen}(\widehat{q}), 0) + 1)/2 \rfloor + 1$ // Bits to drop to reach the standard form
$M \leftarrow \lceil \rho / (\texttt{Zi\_thresh} - 1) \rceil + 4$
$A, B, C, D \leftarrow \mathsf{LehmerGLZi}(2N, X + Y\mathbf{i}, 0, \widehat{q}, M)$
// No need for denominators, scale by $\sim \sqrt{p}$ to use the standard form
$a, c \leftarrow A + \frac{C}{\widehat{q}}\mathbf{j}, B + \frac{D}{\widehat{q}}\mathbf{j}$
$b, d, \leftarrow \mathbf{i}a, \mathbf{i}c$
$B' \leftarrow [a/2, b/2, c/2, d/2]$
return B'

4.3.4.3. Multiplication of inert ideals. As shown in [LS26], the product of inert ideals of coprime degree always has a basis with a very specific shape which can be computed efficiently.

Definition 4.3.5. MulIdeal consists of five integers $I = (N, (X_1, X_2, Y_1, Y_2))$ with $\mathsf{IdealNorm}(I) = N$.

Algorithm 4.73 $\mathsf{MulIdealBasis}(\mathsf{MulIdeal})\colon \mathsf{LatticeBasis}$

$$(N, (X_1, X_2, Y_1, Y_2)) \mapsto \left\langle N, N\mathbf{i}, X_1 + X_2 + \frac{Y_1 - Y_2}{2}\mathbf{i} + \frac{\mathbf{i}+\mathbf{j}}{2}, -\frac{Y_1 + Y_2}{2} + (X_2 - X_1)\mathbf{i} + \frac{1+\mathbf{k}}{2} \right\rangle$$

Algorithm 4.74 IdealMultiplication

Input: $I_1 = (N_1, (x_1, y_1))\colon \mathsf{InertIdeal}$,
$I_2 = (N_2, (x_2, y_2))\colon \mathsf{InertIdeal}$,
Output: $\perp$, if N_1, N_2 are not coprime
$K\colon \mathsf{MulIdeal}$, with $K = \overline{I_1} \cdot I_2$ otherwise

1: $d, u_1, u_2 \leftarrow \mathsf{XGCD}(N_1, N_2)$ // $u_1 N_1 + u_2 N_2 = d$
2: **if** $d \neq 1$ **then**
3: **return** $\perp$
4: $X_1 \leftarrow N_2(-u_2 x_1 \mod (N_1))$
5: $X_2 \leftarrow N_1(u_1 x_2 \mod (N_2))$
6: $Y_1 \leftarrow N_2(-u_2(2y_1 + 1) \mod (2N_1))$
7: $Y_2 \leftarrow N_1(u_1(2y_2 + 1) \mod (2N_2))$
8: **return** $N_1 N_2, (X_1, X_2, Y_1, Y_2)$

Algorithm 4.75 MulIdealDualReduction

Input: $I\colon \mathsf{MulIdeal}$
Output: $\lambda_1, \lambda_2, \lambda_3, \lambda_4\colon \mathsf{Int}$, positive, the squared successive minima of the dual lattice $I^\vee$ of I, scaled by $N^2 p$.
$T \in \mathbb{Z}^{4\times 4}$, such that $T^{-1}(B^\vee)$ is a Minkowski-reduced basis of $I^\vee$, where $B^\vee$ is the reversed dual basis of B.

1: $N, (X_1, X_2, Y_1, Y_2) \leftarrow I$
2: $A, B, C, E \leftarrow 2(X_1 + X_2), Y_1 - Y_2 + 1, 1 - Y_1 - Y_2, 2(X_2 - X_1)$
3: $\lambda_1, \lambda_2, \lambda_3, \lambda_4, T \leftarrow \mathsf{DualLatReduction4DZ}(N, A, B, C, E, \mathsf{BitLenCommitSkChall})$ // β_λ is the bound of Section 4.3.3
4: **return** $(\lambda_1, \lambda_2, \lambda_3, \lambda_4, T)$

4.3.4.4. Generic $\mathcal{O}_0$-ideals. Not all $\mathcal{O}_0$-ideals that we encounter in SQISIGN are going to be inert. Thus, we introduce a new type to handle generic $\mathcal{O}_0$-ideals. The approach taken below is again inspired by [LS26].

Definition 4.3.6. $\mathsf{GenericO_0Ideal}$ consists of a pair (α, I) where $\alpha \in \mathbb{Z}[\mathbf{i}]$ and $I \in \mathsf{InertIdeal}$. If $J = (\alpha, I)$, then the ideal J corresponds to $I \cdot \alpha$.

Algorithm 4.76 GenToIdealOdd

Input: N: Int, positive, odd, and coprime to p
γ: QuatElem, with $\mathrm{nrd}(\gamma) = 0 \mod N$ and $\gamma \in \mathcal{O}_0$
Output: I: GenericO$_0$Ideal, the ideal equal to $\mathcal{O}_0\langle\gamma, N\rangle$

$a, b, c, d, e \leftarrow$ CoordinatesB(ModO$_0(\gamma, N)$)
$u + \mathbf{i}v \leftarrow$ GaussGCD$(a + \mathbf{i}b, c - \mathbf{i}d)$
$u + \mathbf{i}v \leftarrow$ GaussGCD$(u + \mathbf{i}v, N)$
if $u + \mathbf{i}v == \pm i$ **then**
  $u + \mathbf{i}v \leftarrow 1$
$N_2 \leftarrow u^2 + v^2$
$N_1 \leftarrow N/N_2$
$\gamma \leftarrow$ ModO$_0\left(\text{Mul}\left(\gamma, \frac{u-\mathbf{i}v}{N_2}\right), N_1\right)$
$(N_1, (x, y)) \leftarrow$ InertGenToIdealOdd(N_1, γ)
return $u + \mathbf{i}v, (N_1, (x, y))$

Algorithm 4.77 Intersect

Input: $I_1 = (u + \mathbf{i}v), (N_1, (x_1, y_1))$: GenericO$_0$Ideal
$I_2 = (N_2, (x_2, y_2))$: InertIdeal
Output: $\perp$, if $N_1(u^2 + v^2)$ and N_2 are not coprime,
I: InertIdeal, with $(I_1 \cdot (u + \mathbf{i}v)) \cap I_2 = I \cdot (u + \mathbf{i}v)$ otherwise

$X_2 + \mathbf{i}Y_2 \leftarrow$ Mul$(u - \mathbf{i}v, u - \mathbf{i}v)$
$X_2 + \mathbf{i}Y_2 \leftarrow$ ModO$_0$ (Mul$(2x_2 + \mathbf{i}(2y_2 + 1), X_2 + iY_2), N_2)$
// $X_2 + \mathbf{i}Y_2 = (2x_2 + \mathbf{i}(2y_2 + 1))(u - \mathbf{i}v)^2 \mod N_2\mathcal{O}_0$
$t \leftarrow$ ModularInverse$(u^2 + v^2, N_2)$
$X_2 \leftarrow X_2/2$ // X_2 is an integer
$Y_2 \leftarrow (Y_2 - u^2 - v^2)/2$ // Y_2 is an integer
$X_2 \leftarrow X_2 t \mod N_2$
$Y_2 \leftarrow Y_2 t \mod N_2$
$J_2 \leftarrow (N_2, (X_2, Y_2))$
return InertIntersect(I_1, J_2)

4.3.4.5. Equivalent ideals. We first specify an algorithm SampleRandomFromBasis to sample an element from an ideal given a basis of this ideal. This function is used as a subroutine to find equivalent ideals with specific properties.

Algorithm 4.78 SampleRandomFromBasis

Input: $I = (\alpha_1, \alpha_2, \alpha_3, \alpha_4)$: LatticeBasis,
b: Int, positive
prng: PRNG
Output: β: QuatElem, $\beta \in I$

$c_1 \leftarrow$ SampleFromInterval$(-b, b,$ prng$)$
$c_2 \leftarrow$ SampleFromInterval$(-b, b,$ prng$)$
$c_3 \leftarrow$ SampleFromInterval$(-b, b,$ prng$)$
$c_4 \leftarrow$ SampleFromInterval$(-b, b,$ prng$)$
$\beta \leftarrow \sum_{i=1}^{4} c_i \alpha_i$
return β

The algorithms EquivalentCoprimeIdeal and SmallestEquivalentIdeal below use the global parameter EIBox defined in Table 9. The algorithm EquivalentCoprimeIdeal additionally uses EIBoundIteration, defined in the same table. Moreover, we introduce the convention:

$$\mathrm{GCD}(0, N) = 1 \Leftrightarrow \mathrm{IsPrime}(N) = \texttt{true}.$$

Algorithm 4.79 EquivalentCoprimeIdeal

Input: I: InertIdeal,
M: Int, $M \in 2\mathbb{N}$,
B_λ: Int, positive, a bound
prng: PRNG
Output: J: InertIdeal, $J \sim I$ of norm coprime to M if $M \neq 0$ and prime norm otherwise
β: QuatElem, $\beta \in I$ such that $J = \chi_I(\beta)$

$\alpha_1, \alpha_2, \alpha_3, \alpha_4 \leftarrow$ InertIdealReduction(I, B_λ) // The output basis needs to satisfy some specific conditions, the concrete conditions are detailed in the output of LehmerGLZi.
counter $\leftarrow 0$
while counter $\leq$ EIBoundIteration **do**
  counter $\leftarrow$ counter $+ 1$
  $\beta \leftarrow$ SampleRandomFromBasis$((\alpha_1, \alpha_2, \alpha_3, \alpha_4)$, EIBox, prng$)$
  $\beta \leftarrow$ MakePrimitive(β)
  $N' \leftarrow$ Norm(β)/IdealNorm(I)
  if GCD$(N', M) == 1$ **then**
    $J \leftarrow$ InertGenToIdeal$(\overline{\beta}, N')$
    if $J \neq \perp$ **then**
      return J, β
raise Exception ("EquivalentCoprimeIdeal failed: no norm satisfying the constraints was found") // see Section 9.3

Algorithm 4.80 SmallestEquivalentIdeal

Input: I: InertIdeal,
prng$_{\text{def}}$: PRNG
Output: J: InertIdeal, J is the ideal of smallest norm in the equivalence class of I
α_1: QuatElem, $\alpha_1 \in I$ such that $J = \chi_I(\alpha_1)$

$\alpha_1, \alpha_2, \alpha_3, \alpha_4 \leftarrow$ InertIdealReduction(I)
$N \leftarrow$ IdealNorm(I)
$N' \leftarrow$ Norm$(\alpha_1)/N$
while true **do**
  $\beta \leftarrow$ SampleRandomFromBasis$((\alpha_1, \alpha_2, \alpha_3, \alpha_4)$, EIBox, prng$_{\text{def}})$
  $M \leftarrow$ Norm$(\beta)/N$
  if GCD$(N', M) == 1$ **then**
    $\beta \leftarrow$ MakePrimitive(β)
    $\beta \leftarrow$ ModO$_0(\frac{1}{N}$Mul$(\beta, \overline{\alpha_1}), N')$
    $J \leftarrow$ InertGenToIdeal(β, N')
    return J, α_1

4.3.4.6. Sampling algorithms. Finally, we conclude this section with a few functions to sample random ideals, or random elements in ideals.

The algorithm UniformO$_0$ClassSampling implicitly uses a parameter D_{mix} described more precisely in Section 4.7.

Algorithm 4.81 UniformO$_0$ClassSampling

Input: M: Int
prng: PRNG
Output: I: InertIdeal, where the class of I is sampled with small bias among the $\mathcal{O}_0$-ideal classes different from the class of $\mathcal{O}_0$, and IdealNorm(I) is coprime to M if $M \neq 0$ and prime otherwise

$I \leftarrow$ PrimeNormIdealInertSampling$(D_{\text{mix}}$, prng$)$
$I, \_ \leftarrow$ EquivalentCoprimeIdeal$(I, M$, BitLenDegMix, prng$)$
return I

The algorithm RandomIdealGivenNorm implicitly uses a parameter RICofactor described more precisely in Section 4.7.

Algorithm 4.82 RandomIdealGivenNorm

Input: N: Int, positive, odd, and coprime to p
α: QuatElem, with $\alpha \in \mathcal{O}_0$ and Norm(α) coprime to N
prng: PRNG
Output: I: GenericO$_0$Ideal, an $\mathcal{O}_0$-ideal of norm N sampled with small bias

1: $\gamma \leftarrow$ RepresentInteger(RICofactor $\cdot N$, prng)
2: **repeat**
3: $\quad x \leftarrow$ SampleFromInterval$(1, N,$ prng$)$
4: $\quad y \leftarrow$ SampleFromInterval$(1, N,$ prng$)$
5: $\quad z \leftarrow$ SampleFromInterval$(1, N,$ prng$)$
6: $\quad w \leftarrow$ SampleFromInterval$(1, N,$ prng$)$
7: $\quad \beta \leftarrow x + y\mathbf{i} + z\mathbf{j} + w\mathbf{k}$
8: $\quad M \leftarrow \mathrm{nrd}(\beta) \mod N$
9: **until** GCD$(M, N) == 1$
10: $\gamma' \leftarrow$ ModO$_0$ (Mul (γ, β), N)
11: $\gamma' \leftarrow$ ModO$_0$ (Mul (γ', α), N)
12: $(\theta, I) \leftarrow$ GenToIdealOdd(N, γ')
13: **return** (θ, I)

The algorithm RandomBoundedElement implicitly uses the security parameter λ to bound the number of iterations in the sampling loop. According to Section 9.5, we should always be able to find an element of odd norm in the ideal, so the exception from Line 14 should be raised only with negligible probability.

Algorithm 4.83 RandomBoundedElement

Input: I: MulIdeal,
B: Int, positive, a bound.
prng: PRNG
Output: M: Int, odd and smaller than B
α: QuatElem, an element in I sampled with small bias, of norm IdealNorm$(I)M$

1: $\mathbf{b}_1, \mathbf{b}_2, \mathbf{b}_3, \mathbf{b}_4 \leftarrow$ MulIdealBasis(I)
2: $\lambda_1, \lambda_2, \lambda_3, \lambda_4, T \leftarrow$ MulIdealDualReduction(I) // The output matrix T needs to satisfy some specific conditions, the concrete conditions are detailed in the output of MinkowskiReduce.
3: $N \leftarrow$ IdealNorm(I)
4: **for** i **from** 1 **up to** BoundIterationRBE **do**
5: $\quad$ **for** i **from** 1 **up to** 4 **do**
6: $\quad\quad \beta_i \leftarrow \left\lfloor \sqrt{B\lambda_i/(Np)} \right\rfloor$ // $\beta_i = \sqrt{BN\lambda_i(I^\vee)}$
7: $\quad\quad x_i \leftarrow$ SampleFromInterval$(-\beta_i, \beta_i,$ prng$)$
8: $\quad y' \leftarrow T(x_1, x_2, x_3, x_4)^t$ // $y' = (y'_1, y'_2, y'_3, y'_4)^t$
9: $\quad y \leftarrow (y'_4, y'_3, y'_2, y'_1)$ // Account the reverse dual basis
10: $\quad \alpha \leftarrow y_1\mathbf{b}_1 + y_2\mathbf{b}_2 + y_3\mathbf{b}_3, +y_4\mathbf{b}_4$
11: $\quad M \leftarrow$ Norm$(\alpha)/N$ // Equivalently $M = yGy^t$, for G the Gram matrix of $(\mathbf{b}_1, \ldots, \mathbf{b}_4)$ for nrd, divided by N
12: $\quad$ **if** $M \leq B$ **and** $M \bmod 2 == 1$ **then**
13: $\quad\quad$ **return** M, α
14: **raise** Exception ("RandomBoundedElement failed: no odd norm element found of norm $\leq B$") // See Section 9.5

4.4. Elliptic curves

In this section, we describe supporting algorithms required in SQIsign, related to the group law, torsion basis computations, pairings, elliptic-curve discrete logarithms, and one-dimensional isogenies.

4.4.1. x-only arithmetic

Given $P = (X_P : Z_P)$, $Q = (X_Q, Z_Q)$ and $D = (X_D : Z_D)$ with $D = P - Q \neq (1:0), (0:1)$, we can compute $P + Q = S = (X_S : Z_S)$ as

$$\begin{cases} X_S &= Z_D \cdot ((X_P - Z_P)(X_Q + Z_Q) + (X_P + Z_P)(X_Q - Z_Q))^2, \\ Z_S &= X_D \cdot ((X_P - Z_P)(X_Q + Z_Q) - (X_P + Z_P)(X_Q - Z_Q))^2. \end{cases} \tag{6}$$

To double P, e.g., when $P = Q$ and therefore $D = P - Q = (1:0)$, we have

$$\begin{cases} X_{[2]P} &= (X_P + Z_P)^2 (X_P - Z_P)^2, \\ Z_{[2]P} &= (4X_P Z_P)\left((X_P - Z_P)^2 + ((A+2)/4(4X_P Z_P))\right). \end{cases} \tag{7}$$

When representing A as a projective coordinate $(A : C)$, we define $(A_{24} : C_{24})$ by $A_{24} = A + 2C$ and $C_{24} = 4C$, which represents the value $(A + 2)/4$ used in pseudo-doubling. The function CurveToMontgomeryA computes the A coefficient of a Montgomery curve.

$$\mathsf{CurveToMontgomeryA}(A : C) \mapsto \frac{A}{C}$$

We define three algorithms for the arithmetic on Montgomery curves:

- xDBL takes as input an ECPoint $(X_P : Z_P)$ representing P and a ECCurve $(A_{24} : C_{24})$ representing E_A, and outputs a ECPoint $(X_{[2]P} : Z_{[2]P})$ representing $[2]P$.
- xADD takes as input three ECPoint values (X_P, Z_P), (X_Q, Z_Q), and (X_{P-Q}, Z_{P-Q}) representing P, Q, and $P - Q$, and outputs the ECPoint (X_{P+Q}, Z_{P+Q}) representing $P + Q$.
- xDBLADD efficiently combines xDBL and xADD, taking as input three ECPoint values (X_P, Z_P), (X_Q, Z_Q), and (X_{P-Q}, Z_{P-Q}) representing P, Q, and $P - Q$, and an ECCurve $(A_{24} : C_{24})$ representing E_A, and outputs the ECPoint values (X_{P+Q}, Z_{P+Q}) and $(X_{[2]P}, Z_{[2]P})$ representing the points $P + Q$ and $[2]P$, respectively.

Algorithm 4.84 xDBL$(P, (A_{24} : C_{24}))$ Cost: $2\mathbf{S} + 4\mathbf{M} + 4\mathbf{a}$

Input: $P = (X_P : Z_P)$: ECPoint
$(A_{24} : C_{24})$: ECCurve
Output: $[2]P = (X_{2P} : Z_{2P})$: ECPoint

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** $(X_{2P} : Z_{2P})$

Algorithm 4.85 xADD$(P, Q, P - Q)$ Cost: $2\mathbf{S} + 4\mathbf{M} + 6\mathbf{a}$

Input: $P = (X_P : Z_P)$: ECPoint,
$Q = (X_Q : Z_Q)$: ECPoint,
$P - Q = (X_{P-Q} : Z_{P-Q})$: ECPoint
Output: $P + Q = (X_{P+Q} : Z_{P+Q})$: ECPoint

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** $(X_{P+Q} : Z_{P+Q})$

Algorithm 4.86 xDBLADD$(P, Q, P-Q, (A_{24} : C_{24}))$ Cost: $4\mathbf{S} + 8\mathbf{M} + 8\mathbf{a}$

Input: $P = (X_P : Z_P)$: ECPoint,
$Q = (X_Q : Z_Q)$: ECPoint,
$P - Q = (X_{P-Q} : Z_{P-Q})$: ECPoint,
$(A_{24} : C_{24})$: ECCurve
Output: $[2]P = (X_{2P} : Z_{2P})$: ECPoint,
$P + Q = (X_{P+Q} : Z_{P+Q})$: ECPoint

1: $t_0 \leftarrow X_P + Z_P$
2: $t_1 \leftarrow X_P - Z_P$
3: $X_{2P} \leftarrow t_0^2$
4: $t_2 \leftarrow X_Q - Z_Q$
5: $X_{P+Q} \leftarrow X_Q + Z_Q$
6: $t_0 \leftarrow t_0 \cdot t_2$
7: $Z_{2P} \leftarrow t_1^2$
8: $t_1 \leftarrow t_1 \cdot X_{P+Q}$
9: $t_2 \leftarrow X_{2P} - Z_{2P}$
10: $Z_{2P} \leftarrow Z_{2P} \cdot C_{24}$
11: $X_{2P} \leftarrow X_{2P} \cdot Z_{2P}$
12: $X_{P+Q} \leftarrow A_{24} \cdot t_2$
13: $Z_{P+Q} \leftarrow t_0 - t_1$
14: $Z_{2P} \leftarrow Z_{2P} + X_{P+Q}$
15: $X_{P+Q} \leftarrow t_0 + t_1$
16: $Z_{2P} \leftarrow Z_{2P} \cdot t_2$
17: $Z_{P+Q} \leftarrow Z_{P+Q}^2$
18: $X_{P+Q} \leftarrow X_{P+Q}^2$
19: $Z_{P+Q} \leftarrow Z_{P+Q} \cdot X_{P-Q}$
20: $X_{P+Q} \leftarrow X_{P+Q} \cdot Z_{P-Q}$
21: **return** $([2]P, P + Q)$

4.4.1.1. Montgomery ladder. Using the basic operations xDBL, xADD, xDBLADD, we can describe the following x-only variants of scalar multiplications:

- Ladder takes as input the point P: ECPoint, the curve $(A_{24} : C_{24})$: ECCurve, and a positive scalar m, and outputs the point $[m]P$: ECPoint.
- Ladder3pt takes as input points $P, Q, P - Q$: ECPoint, the curve $(A_{24} : C_{24})$: ECCurve, and a positive scalar m, and outputs the point $P + [m]Q$: ECPoint.
- LadderBiscalar takes as input points $P, Q, P - Q$: ECPoint, the curve $(A_{24} : C_{24})$: ECCurve, and positive scalars m and n, and outputs the point $[m]P + [n]Q$: ECPoint. It implements BiscalarMul.
- GluingLadder takes as input points $P, Q, P-Q$: ECPoint, the normalized curve $(A_{24}/C_{24} : 1)$: ECCurve, and a positive integer $e \leq f$, and outputs the points $[2^e - 1]P, [2^e]P$: ECPoint. This ladder is a specific variant used in the HD module.

Algorithm 4.87 Ladder(P, E, m)

Input: $P = (X_P : Z_P)$: ECPoint,
$E = (A_{24} : C_{24})$: ECCurve,
a positive scalar m with binary representation $m = (m_{k-1}, \ldots, m_0)_2$
Output: $[m]P = (X_{[m]P} : Z_{[m]P})$: ECPoint

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

Algorithm 4.88 Ladder3pt$(P, Q, P-Q, E, m)$

Input: $P = (X_P : Z_P)$: ECPoint,
$Q = (X_Q : Z_Q)$: ECPoint,
$P - Q = (X_{P-Q} : Z_{P-Q})$: ECPoint,
$E = (A_{24} : C_{24})$: ECCurve,
a positive scalar m with binary representation $m = (m_{k-1}, \ldots, m_0)_2$
Output: $P + [m]Q = (X_{P+[m]Q} : Z_{P+[m]Q})$: ECPoint

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

Algorithm 4.89 LadderBiscalar$(P, Q, P-Q, E, m, n)$

Input: $P = (X_P : Z_P)$: ECPoint,
$Q = (X_Q : Z_Q)$: ECPoint,
$P - Q = (X_{P-Q} : Z_{P-Q})$: ECPoint,
$E = (A_{24} : C_{24})$: ECCurve,
a positive scalar m with binary representation $m = (m_{k-1}, \ldots, m_0)_2$,
a positive scalar n with binary representation $n = (n_{k-1}, \ldots, n_0)_2$
Output: $[m]P + [n]Q = (X_{[m]P+[n]Q} : Z_{[m]P+[n]Q})$: ECPoint

Recoding stage:
if m is even **and** n is odd **then**
  $(\sigma_0, \sigma_1) \leftarrow (1, 0)$
else
  $(\sigma_0, \sigma_1) \leftarrow (0, 1)$
$m' \leftarrow m, n' \leftarrow n$
if m is even **then**
  $m' \leftarrow m' - 1$
if n is even **then**
  $n' \leftarrow n' - 1$
$b_0 \leftarrow (0, m'_{k-1}, \ldots, m'_0)$
$b_1 \leftarrow (0, n'_{k-1}, \ldots, n'_0)$
$b \leftarrow (b_0, b_1)$
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:
$R_0 \leftarrow (1 : 0)$
$(T_0, T_1) \leftarrow (P, Q)$
$R_1 \leftarrow T_{\sigma_0}$
$R_2 \leftarrow T_{\sigma_0+1 \pmod 2}$
$D_1 \leftarrow R_1$
$D_2 \leftarrow R_2$
$R_2 \leftarrow$ 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**
  $h \leftarrow r_{2i} + r_{2i+1}$
  $T_0 \leftarrow R_{h \pmod 2}$
  $(T_0, T_1) \leftarrow (T_0, R_2)$
  $T_0 \leftarrow$ 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**
    $TMP \leftarrow D_1$
    $D_1 \leftarrow D_2$
    $D_2 \leftarrow TMP$
  $T_1 \leftarrow$ xADD(T_1, T_2, D_1)
  $T_2 \leftarrow$ xADD(R_0, R_2, F_1)
  if $h \pmod 2 == 1$ **then**
    $TMP \leftarrow F_1$
    $F_1 \leftarrow F_2$
    $F_2 \leftarrow TMP$
  $R_0 \leftarrow T_0$
  $R_1 \leftarrow T_1$
  $R_2 \leftarrow T_2$
$(X_{[m]P+[n]Q} : Z_{[m]P+[n]Q}) \leftarrow R_{(m \pmod 2)+(n \pmod 2)}$
return $(X_{[m]P+[n]Q} : Z_{[m]P+[n]Q})$

Algorithm 4.90 GluingLadder$(P, Q, P-Q, E, e)$

Input: $P = (X_P : Z_P)$: ECPoint,
$Q = (X_Q : Z_Q)$: ECPoint,
$P - Q = (X_{P-Q} : Z_{P-Q})$: ECPoint,
$E = (A_{24}/C_{24} : 1)$: ECCurve, normalized,
a positive integer $e \leq f$
Output: $[2^e - 1]P$: ECPoint,
$[2^e]P$: ECPoint

$(R_0, R_1) \leftarrow ((X_Q : Z_Q), (X_P : Z_P))$
for i **from** 1 **up to** e **do**
  $(R_1, R_0) \leftarrow$ xDBLADD$(R_1, R_0, P - Q, E)$
return (R_1, R_0)

4.4.1.2. Point difference. To deterministically compute either one of $P + Q$ or $P - Q$ given only P and Q, we use [RS17, Prop. 3] to compute the set $\{x_{P-Q}, x_{P+Q}\}$. These values can be described in terms of the classical biquadratics:

$$x_{P \pm Q} = \frac{B_{XZ} \pm \sqrt{B_{XZ}^2 - B_{ZZ} B_{XX}}}{B_{ZZ}}, \tag{8}$$

where

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

These values are computed using Biquadratics.

Algorithm 4.91 Biquadratics(E, P_1, P_2)

Input: A curve $E = (A : C)$: ECCurve,
$P_1 = (X_{P_1} : Z_{P_1})$: ECPoint,
$P_2 = (X_{P_2} : Z_{P_2})$: ECPoint
Output: B_{XX}, B_{XZ}, B_{ZZ}: Fp2Elem, as defined in Equation (8)

$B_{XX} \leftarrow C \cdot (X_{P_1} X_{P_2} - Z_{P_1} Z_{P_2})^2$
$B_{XZ} \leftarrow C \cdot (X_{P_1} X_{P_2} + Z_{P_1} Z_{P_2})(X_{P_1} Z_{P_2} + Z_{P_1} X_{P_2}) + 2A X_{P_1} X_{P_2} Z_{P_1} Z_{P_2}$
$B_{ZZ} \leftarrow C \cdot (X_{P_1} Z_{P_2} - Z_{P_1} X_{P_2})^2$
return B_{XX}, B_{XZ}, B_{ZZ}

For SQIsign, it is important to always obtain the same deterministic result, which requires a deterministic choice of square root and solution to Eq. (8), independent of projective representation of P and Q. We describe the algorithm to achieve this in ProjectiveDifference.

Algorithm 4.92 ProjectiveDifference(P, Q, E)

Input: $P = (X_P : Z_P)$: ECPoint,
$Q = (X_Q : Z_Q)$: ECPoint,
$E = (A : C)$: ECCurve
Output: A deterministic $(X_{PQ} : Z_{PQ})$: ECPoint, either x_{P-Q} or x_{P+Q}

$B_{XX} \leftarrow C \cdot (X_P X_Q - Z_P Z_Q)^2$
$B_{XZ} \leftarrow C \cdot (X_P X_Q + Z_P Z_Q)(X_P Z_Q + Z_P X_Q) + 2A X_P X_Q Z_P Z_Q$
$B_{ZZ} \leftarrow C \cdot (X_P Z_Q - Z_P X_Q)^2$
$\gamma \leftarrow C \cdot (\overline{C} \cdot \overline{X_P Z_Q - Z_P X_Q})^2$
$B_{XX} \leftarrow \gamma \cdot B_{XX}$, $B_{XZ} \leftarrow \gamma \cdot B_{XZ}$, $B_{ZZ} \leftarrow \gamma \cdot B_{ZZ}$,
$\delta \leftarrow$ Fp2SQRT$(B_{XZ}^2 - B_{XX} B_{ZZ})$
$X_{PQ} \leftarrow B_{XZ} - \delta$, $Z_{PQ} \leftarrow B_{ZZ}$
return $(X_{PQ} : Z_{PQ})$

Beyond ProjectiveDifference, which only recovers a deterministic choice out of $\{x_{P-Q}, x_{P+Q}\}$, we require a function that returns the specific difference $x_{P_1-P_2}$ for a given pair of points. We compute it in SharedDifference from the two points P_1, P_2 together with their translates P_1+Q and P_2+Q by a common auxiliary point Q. Given such inputs, we do not require an additional square root: the value $x_{P_1-P_2}$ is a root for the quadratics from Eq. (8) associated to (P_1, P_2) as well as (P_1+Q, P_2+Q), and so we may simply compute this single shared root. The other root of each quadratic is $x_{P_1+P_2}$ and $x_{P_1-P_2+2Q}$ respectively, which differ. Writing (B_{XX}, B_{XZ}, B_{ZZ}) and $(B'_{XX}, B'_{XZ}, B'_{ZZ})$ for the biquadratic coefficients of these two pairs, the common root is

$$x_{P_1-P_2} = \frac{2(B'_{XZ}B_{XX} - B_{XZ}B'_{XX})}{B'_{ZZ}B_{XX} - B_{ZZ}B'_{XX}}.$$

Algorithm 4.93 SharedDifference$(E, P_1, P_2, P_1+Q, P_2+Q)$

Input: A curve $E = (A : C)$: ECCurve,
$P_1 = (X_{P_1} : Z_{P_1})$: ECPoint,
$P_2 = (X_{P_2} : Z_{P_2})$: ECPoint,
$P_1 + Q = (X_{P_1+Q} : Z_{P_1+Q})$: ECPoint,
$P_2 + Q = (X_{P_2+Q} : Z_{P_2+Q})$: ECPoint
Output: $(X_{P_1-P_2} : Z_{P_1-P_2})$: ECPoint, the x-coordinate of $P_1 - P_2$

1: $B_{XX}, B_{XZ}, B_{ZZ} \leftarrow$ Biquadratics(E, P_1, P_2)
2: $B'_{XX}, B'_{XZ}, B'_{ZZ} \leftarrow$ Biquadratics(E, P_1+Q, P_2+Q)
3: $X_{P_1-P_2} \leftarrow 2(B'_{XZ} \cdot B_{XX} - B_{XZ} \cdot B'_{XX})$
4: $Z_{P_1-P_2} \leftarrow B'_{ZZ} \cdot B_{XX} - B_{ZZ} \cdot B'_{XX}$
5: **return** $(X_{P_1-P_2} : Z_{P_1-P_2})$

4.4.2. Torsion bases

SQIsign represents a torsion basis (R, S) for $E_{A/C}[N] \cong \mathbb{Z}_N \times \mathbb{Z}_N$, for N coprime to p, using x-only coordinates. This requires three ECPoint values, namely x_R, x_S, and the x-coordinate of their difference, x_{R-S}. This defines the basis (R, S) up to a sign, e.g., $(-R, -S)$ has the same representation.

Definition 4.4.1. The type ECBasis$_e$ represents a torsion basis (R, S) for $E_{A/C}[2^e]$ by the triple of points

- $R = (X_R : Z_R)$: ECPoint,
- $S = (X_S : Z_S)$: ECPoint, and
- $RS = (X_{R-S} : Z_{R-S})$: ECPoint.

SQIsign requires torsion bases for 2^e with $e \leq f$. Such torsion bases need to be constructed deterministically, as they encode the kernels of crucial isogenies, such as φ_{chl}. We describe the deterministic generation of a basis (R, S) for $E[2^e]$ for any curve other than E_0 in TorsionBasisToHint, using a modified version of [ZSP+18, Alg. 3.1]. This algorithm requires a normalized ECCurve $(A : 1)$ and computes the quadratic residuosity of A to ensure that $x_R \leftarrow -A/(1 + i \cdot h)$ is a non-square in $\mathbb{F}_{p^2}$, for a square h. 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 $h \in \mathbb{F}_p$ is always a square in $\mathbb{F}_{p^2}$, hence when A is non-square, any $\mathbb{F}_p$-multiple $h \cdot A$ is of the right form for x_R.

Sampling such a torsion basis for $E[2^e]$ 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_R. Both of these can be provided in the signature as *hints*: the signer uses TorsionBasisToHint, which generates not only a basis, but also 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}).

It can happen that TorsionBasisFromHint needs a basis of order 2^{e_h} with $e_h < e$. If (x_R, x_S, x_{RS}) is the basis of $E[2^e]$ returned by TorsionBasisToHint$(A, e, 0)$, then TorsionBasisFromHint(A, e_h) makes a deterministic choice of $[2^{e-e_h}](R \pm S)$ by calling ProjectiveDifference with input $[2^{e-e_h}]R$, $[2^{e-e_h}]S$ and E. This choice might not equal $[2^{e-e_h}]RS$. To account for this, TorsionBasisToHint takes an extra parameter e_h, which, when nonzero, is used to adapt the returned basis of order 2^e from TorsionBasisToHint(A, e, e_h) to be compatible with a call of TorsionBasisFromHint(A, e_h);

Algorithm 4.94 TorsionBasisToHint(A, e, e_h)

Input: $(A : 1)$: ECCurve (normalized),
two integers $e_h < e \leq f$.
Output: (R, S, RS): ECBasis$_e$,
hint: Hint$_{e-e_h}$

if A is a square **then**
  $h_A \leftarrow 1$
  $h \leftarrow 0$
  repeat
    repeat
      $h \leftarrow h + 1$
    until $(1 + h^2)$ is **not** a square in $\mathbb{F}_p$
    $R \leftarrow (-A : 1 + i \cdot h)$
  until $R \in E_A(\mathbb{F}_{p^2})$
else
  $h_A \leftarrow 0$
  $h \leftarrow 0$
  repeat
    $h \leftarrow h + 1$,
    $R \leftarrow (h \cdot A : 1)$
  until $R \in E_A(\mathbb{F}_{p^2})$
$S \leftarrow (-(X_R + AZ_R) : Z_R)$
$R \leftarrow$ Ladder$(R, (A + 2 : 4), \frac{p+1}{2^e})$
$S \leftarrow$ Ladder$(S, (A + 2 : 4), \frac{p+1}{2^e})$
$RS \leftarrow$ ProjectiveDifference$(R, S, (A : 1))$
if $e_h > 0$ **then**
  $RS_2 \leftarrow$ xADD(R, S, RS)
  $T_1 \leftarrow$ Ladder$(RS, ((A + 2)/4, 1), 2^{e-e_h})$
  $T_2 \leftarrow$ Ladder$(RS_2, ((A + 2)/4, 1), 2^{e-e_h})$
  $Z \leftarrow (X_{T_1} Z_{T_2} - Z_{T_1} X_{T_2}) \cdot \overline{Z_{T_1} Z_{T_2}} \cdot Z_{T_1} \overline{Z}_{T_1} \cdot Z_{T_2} \overline{Z}_{T_2}$
  if Fp2SQRT$(Z^2) = Z$ **then** $RS \leftarrow RS_2$
if $h \geq 128$ **then**
  hint $\leftarrow 0$
else
  hint $\leftarrow 2h + h_A$
$S, RS \leftarrow RS, S$
return (R, S, RS) and hint

Algorithm 4.95 TorsionBasisFromHint(E, e, hint)

Input: $E = (A : 1)$: ECCurve (normalized),
an integer $e \leq f$,
hint: Hint$_e$
Output: (R, S, RS): ECBasis$_e$

if hint $== 0$ **then**
  $\mathcal{B}$, hint $\leftarrow$ TorsionBasisToHint$(A, e, 0)$
  return $\mathcal{B}$
$h_A \leftarrow$ hint $\mod 2$
$h \leftarrow$ hint $\div 2$
if $h_A == 1$ **then**
  $R \leftarrow (-A : 1 + i \cdot h)$
else
  $R \leftarrow (h \cdot A : 1)$
$S \leftarrow (-(X_R + AZ_R) : Z_R)$
$R \leftarrow$ Ladder$(R, (A + 2 : 4), \frac{p+1}{2^e})$
$S \leftarrow$ Ladder$(S, (A + 2 : 4), \frac{p+1}{2^e})$
$RS \leftarrow$ ProjectiveDifference$(R, S, (A : 1))$
$S, RS \leftarrow RS, S$
return (R, S, RS)

One can show that for a basis (R, S) constructed using TorsionBasisToHint, we have that $[2^{e-1}](R + S) = (0, 0)$, which is a property we assume in later algorithms when computing isogenies. To explicitly have a basis point with this property, 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).

Note that the computation of x_{RS} given x_R and x_S must be done deterministically, and any implementation must make the same deterministic choice as the reference implementation, which uses ProjectiveDifference. Furthermore, 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 is verified whenever they are used, which implicitly ensures they lie on E. The reference implementation does this in TwoIsogenyChain and Section 4.5.

4.4.3. One-dimensional isogenies

SQIsign only requires one-dimensional isogenies of degree 2^e, which we compute through chains of 4-isogenies. We use the kernel representation to represent such isogenies.

Definition 4.4.2. The type ECIsog represents an isogeny of degree 2^e by

- its domain curve $(A : C)$: ECCurve,
- a kernel generator K: ECPoint, and
- an even integer e, double the length of the chain of isogenies of degree 4.

An isogeny φ of degree 2^e is computed as a chain of isogenies of degree 4, which relies on FourIsogenyCodomain. We also require evaluation of points $P \mapsto \varphi(P)$, which relies on FourIsogenyEval. Algorithmically, we get the following.

- FourIsogenyCodomain takes as input a kernel generator P: ECPoint of order 4 with $[2]P \neq (0 : 1)$, and outputs the codomain $(A'_{24} : C'_{24})$: ECCurve of the 4-isogeny $\varphi\colon E \to E/\langle P\rangle$, and constants `consts` needed for FourIsogenyEval.
- FourIsogenyEval takes as input a point Q: ECPoint, and the constants `consts` returned by FourIsogenyCodomain, and outputs the image point $\varphi(Q)$: ECPoint.
- TwoIsogenyChain takes as input a kernel generator P: ECPoint of order 2^{e+2}, the curve $(A'_{24} : C'_{24})$: ECCurve, the positive even integer e, and a list of points `pts` $= [Q$: ECPoint$]$ to be evaluated, and outputs the codomain curve $(A'_{24}$: $C'_{24})$: ECCurve of the 2^e-isogeny $\varphi\colon E \to E/\langle[4]P\rangle$ together with the list of evaluated points `pts`$' = [\varphi(Q)\colon Q \in$ `pts`$]$.

When computing 2^e-isogenies from a given point $P \in E$ of order 2^{e+2}, 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 in the first isogeny step of the computation of this chain. Furthermore, for honestly generated inputs, our implementation choices guarantee that $[2]P = (0 : 1)$ does not occur for P an input to FourIsogenyCodomain. Therefore, TwoIsogenyChain raises an error on such inputs and verification rejects signatures that lead to these cases.

To ensure a canonical choice between the six possible Montgomery coefficients for an elliptic curve, we choose the largest A out of these six. This algorithm, denoted NormalizeMontgomery, takes a 4-torsion point $Q = (X : Z)$: ECPoint, on a Montgomery curve E, computes the six possible Montgomery coefficients $A_0, \ldots, A_5$, and returns $(\max A_i : 1)$: ECCurve.

Algorithm 4.96 NormalizeMontgomery

Input: $P = (X_P\colon Z_P)$: ECPoint of order 4 on a Montgomery curve, with $P \neq (\pm 1\colon 1)$
Output: $(A_{\max} : 1)$: ECCurve, a normalized canonical Montgomery coefficient of the curve.

1: $a, b \leftarrow X_P + Z_P, X_P - Z_P$ // $a, b \neq 0$
2: $a' \leftarrow \mathbf{i}a$
3: $b' \leftarrow \mathbf{i}b$
4: $a_1 \leftarrow a + b$
5: $b_1 \leftarrow a - b$
6: $a_2 \leftarrow a + b'$ // $a_2 = a + \mathbf{i}b$
7: $b_2 \leftarrow a - b'$ // $b_2 = a - \mathbf{i}b$
8: $(A_0, C_0) \leftarrow$ ComputeMontCoeff(a, b)
9: $(A_1, C_1) \leftarrow$ ComputeMontCoeff(a_1, b_1)
10: $(A_2, C_2) \leftarrow$ ComputeMontCoeff(a_2, b_2)
11: $C_0, C_1, C_2 \leftarrow$ Fp2BatchInverse$([C_0, C_1, C_2])$
12: **for** i **from** 0 **up to** 2 **do**
13: $\quad A_i \leftarrow A_i \cdot C_i$
14: $\quad A_{i+3} \leftarrow -A_i$
15: $A_{\max} \leftarrow A_0$
16: **for** i **from** 1 **up to** 5 **do**
17: $\quad$ **if** Fp2Compare $(A_{\max}, A_i)$ **then**
18: $\quad\quad A_{\max} \leftarrow A_i$
19: **return** $(A_{\max} : 1)$

Algorithm 4.97 FourIsogenyCodomain(P)

Input: $P = (X_P : Z_P)$: ECPoint of order 4 such that $[2]P \neq (0 : 1)$
Output: $(A'_{24} : C'_{24})$: ECCurve,
and constants `consts`

1: $c_0 \leftarrow Z_P^2$
2: $c_1 \leftarrow X_P - Z_P$
3: $c_2 \leftarrow X_P + Z_P$
4: $t_0 \leftarrow X_P^2$
5: $t_3 \leftarrow c_0 + t_0$
6: $t_4 \leftarrow c_0 - t_0$
7: $A'_{24} \leftarrow t_3 \cdot t_4$
8: $C'_{24} \leftarrow c_0^2$
9: $c_0 \leftarrow c_0 + c_0$
10: $c_0 \leftarrow c_0 + c_0$
11: `consts` $\leftarrow (c_0, c_1, c_2)$
12: **return** $(A'_{24} : C'_{24})$, `consts`

Algorithm 4.98 FourIsogenyEval$(Q,$ `consts`$)$

Input: $Q = (X_Q : Z_Q)$: ECPoint,
and constants `consts` returned by FourIsogenyCodomain
Output: $Q' = (X_{Q'} : Z_{Q'})$: ECPoint

1: $c_0, c_1, c_2 \leftarrow$ `consts`
2: $t_0 \leftarrow X_Q + Z_Q$
3: $t_1 \leftarrow X_Q - Z_Q$
4: $X_{Q'} \leftarrow t_0 \cdot c_1$
5: $Z_{Q'} \leftarrow t_1 \cdot c_2$
6: $t_0 \leftarrow t_0 \cdot t_1$
7: $t_0 \leftarrow t_0 \cdot c_0$
8: $t_1 \leftarrow X_{Q'} + Z_{Q'}$
9: $Z_{Q'} \leftarrow X_{Q'} - Z_{Q'}$
10: $t_1 \leftarrow t_1^2$
11: $Z_{Q'} \leftarrow Z_{Q'}^2$
12: $X_{Q'} \leftarrow t_0 + t_1$
13: $t_0 \leftarrow t_0 - Z_{Q'}$
14: $X_{Q'} \leftarrow X_{Q'} \cdot t_1$
15: $Z_{Q'} \leftarrow Z_{Q'} \cdot t_0$
16: **return** $(X_{Q'} : Z_{Q'})$

Algorithm 4.99 TwoIsogenyChain$(P, E, e, \texttt{pts})$

Input: $P = (X_P : Z_P)$: ECPoint of order 2^{e+2},
$E = (A_{24} : C_{24})$: ECCurve,
a positive even integer $e \leq f$,
and a list of points $\texttt{pts} = [Q$: ECPoint$]$
Output: $(A'_{24} : C'_{24})$: ECCurve,
and a list of evaluated points $\texttt{pts}' = [\varphi(Q) : Q \in \texttt{pts}]$

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

4.4.4. Pairings

The *reduced* Tate-Lichtenbaum pairing of level n, hereafter referred to as the Tate pairing, is a bilinear pairing

$$t_n : E(\mathbb{F}_q)[n] \times E(\mathbb{F}_q)/nE(\mathbb{F}_q) \to \mu_n.$$

We only use Tate pairings of degree $n = 2^e$ with $e \leq f$. As $2^f \mid p + 1$, such pairings are non-degenerate. We first describe how we compute the pairing t_{2^e} using x-only coordinates, before we specify how we use such pairings t_{2^e}.

4.4.4.1. Pairing computation. We compute Tate pairings of degree 2^e for $e \leq f$ in the algorithm Tate, which relies on cubical arithmetic [PRR+25; Rob24]. For points represented as the type ECPoint, this requires cubical addition CubicalDiffAdd and cubical doubling CubicalDbl very similar to xADD and xDBL, resulting in a cubical ladder CubicalLadder very similar to Ladder. However, because we use x-only arithmetic and compute pairings of even degree, we additionally require CubicalTranslate.

Algorithm 4.100 CubicalDiffAdd$(P, Q, \mathrm{inv}(x_{P-Q}))$

Input: $P = (X_P : Z_P)$: ECPoint,
$Q = (X_Q : Z_Q)$: ECPoint,
and the inverse $\mathrm{inv}(x_{P-Q})$: Fp2Elem of x_{P-Q}
Output: $(X_{P+Q} : Z_{P+Q})$: ECPoint

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: $X_{P+Q} \leftarrow (a \cdot d + b \cdot c)^2$
6: $Z_{P+Q} \leftarrow (a \cdot d - b \cdot c)^2$
7: $X_{P+Q} \leftarrow X_{P+Q} \cdot \mathrm{inv}(x_{P-Q})$
8: **return** $(X_{P+Q} : Z_{P+Q})$

Algorithm 4.101 CubicalDbl(E, P)

Input: $E = (A_{24} : C_{24})$: ECCurve,
$P = (X_P : Z_P)$: ECPoint
Output: $[2]P = (X_2 : Z_2)$: ECPoint

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: $Z_2 \leftarrow c \cdot (b + A_{24}/C_{24} \cdot c)$
6: **return** $(X_2 : Z_2)$

Algorithm 4.102 CubicalLadder(E, e, PQ, P, x_Q)

Input: $E = (A_{24} : C_{24})$: ECCurve,
an integer $e \leq f$,
$PQ = (X_{P+Q} : Z_{P+Q})$: ECPoint,
$P = (X_P : Z_P)$: ECPoint,
and x_Q: Fp2Elem
Output: A pair ($[2^e]P$: ECPoint, $[2^e]PQ$: ECPoint)

1: $nPQ \leftarrow PQ$
2: $nP \leftarrow P$
3: $\mathrm{inv}(x_Q) \leftarrow x_Q^{-1}$
4: **for** k **from** 1 **up to** e **do**
5: $\quad nPQ \leftarrow$ CubicalDiffAdd$(nPQ, nP, \mathrm{inv}(x_Q))$
6: $\quad nP \leftarrow$ CubicalDbl(E, nP)
7: **return** (nP, nPQ)

Algorithm 4.103 CubicalTranslate(P, T)

Input: $P = (X_P : Z_P)$: ECPoint,
$T = (X_T : Z_T)$: ECPoint of order 2
Output: $PT = (X(P+T), Z(P+T))$: ECPoint

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)

Note that CubicalDiffAdd as described is off by a factor 4 from the correct cubical arithmetic described in [Rob24]: the algorithm should return $(X_{P+Q}/4, Z_{P+Q}/4)$. However, this does not impact the final result thanks to the final exponentiation for the Tate pairing $t_{2^e}(P, Q)$ over $\mathbb{F}_{p^2}$. However, one should be careful that the current version will not give the correct Tate pairing over $\mathbb{F}_p$. Furthermore, one may call Tate with $(X_{P-Q} : 1)$ instead of $(X_{P+Q} : 1)$ as the third ECPoint. This inverts the final result, and is used in some calls to Tate in SQIsign.

Algorithm 4.104 Tate(E, e, P, Q, PQ)

Input: $E = (A_{24} : C_{24})$: ECCurve,
an integer $e \leq f$,
$P = (X_P : 1)$: ECPoint, normalized, with $[2^e]P = 0_E$,
$Q = (X_Q : 1)$: ECPoint, normalized,
$PQ = (X_{P+Q} : 1)$: ECPoint, normalized
Output: The reduced Tate pairing $t_{2^e}(P, Q) \in \mu_{2^e}$

$nP, nPQ \leftarrow$ CubicalLadder$(E, e-1, PQ, P, X_Q)$
$(X_{nPQ} : Z_{nPQ}) \leftarrow$ CubicalTranslate(nPQ, nP)
$(X_{nP} : Z_{nP}) \leftarrow$ CubicalTranslate(nP, nP)
$\lambda \leftarrow Z_{nPQ}/X_{nP}$
return $\lambda^{(p^2-1)/2^e}$

4.4.4.2. Uses of pairings. In SQIsign, we use the Tate pairing to solve elliptic curve discrete logarithms faster, which is used to compute the matrix M that changes one torsion basis into another: If (P, Q) is a basis for $E[2^f]$, so that $\zeta = t_{2^f}(P, Q)$ is a primitive 2^f-th root of unity, and (R, S) is a basis for $E[2^e]$, then the Tate pairing allows us to compute the change of basis matrix such that

$$(P, Q)\begin{pmatrix} x_1 & x_2 \\ x_3 & x_4 \end{pmatrix} = (R, S)$$

as defined in Eq. (2). The algorithmic description is given in ChangeOfBasis, which requires several x-coordinates of difference points in order to apply Tate using cubical arithmetic, collected in the list `diffs`. To compute these efficiently, we use DiffPoints, which combines (a variant of) ProjectiveDifference to compute $R - P = (X_{R-P} : Z_{R-P})$ together with three calls to SharedDifference to derive $R - Q$, $S - P$, and $S - Q$.

Algorithm 4.105 DiffPoints$(E, (P, Q, P-Q), (R, S, R-S))$

Input: A curve $E = (A : C)$: ECCurve,
a basis $B_1 = (P, Q, P-Q)$: ECBasis$_f$, normalized, for $E[2^f]$,
a basis $B_2 = (R, S, R-S)$: ECBasis$_e$, normalized, for $E[2^e]$
Output: The differences $R-P, S-P, R-Q, S-Q$: ECPoint

$B_{XX}, B_{XZ}, B_{ZZ} \leftarrow$ Biquadratics(E, R, P)
$X_{R-P} \leftarrow B_{XZ} + \sqrt{B_{XZ}^2 - B_{XX}B_{ZZ}}$
$Z_{R-P} \leftarrow B_{ZZ}$
$R - P \leftarrow (X_{R-P} : Z_{R-P})$
$S - P \leftarrow$ SharedDifference$(E, S, P, R-P, R-S)$
$R - Q \leftarrow$ SharedDifference$(E, R, Q, R-P, P-Q)$
$S - Q \leftarrow$ SharedDifference$(E, S, Q, S-P, P-Q)$
return $R-P, S-P, R-Q, S-Q$

Note that if we want to compute a matrix that changes a basis (R, S) of $E[2^e]$ into the basis $([2^{f-e}]P, [2^{f-e}]Q)$, we apply ChangeOfBasis on (P, Q) and (R, S) and invert the resulting matrix.

4.4.5. Barycentric coordinates

Definition 4.4.3. The type BaryCoords represents *barycentric coordinates* $(u : v : w)$ of elliptic curve points P, Q such that $P - Q = (u + v : w)$ and $P + Q = (u - v : w)$.

Algorithm 4.106 ChangeOfBasis$(E, e, (P, Q, P-Q), (R, S, R-S))$

Input: A curve $E = (A_{24} : C_{24})$: ECCurve,
an integer $e \leq f$,
a basis $\mathcal{B}_1 = (P, Q, P-Q)$: ECBasis$_f$, normalized, for $E[2^f]$,
a basis $\mathcal{B}_2 = (R, S, R-S)$: ECBasis$_e$, normalized, for $E[2^e]$
Output: A matrix $\mathbf{M} = (x_i)$, with $1 \leq i \leq 4$, so that $\mathcal{B}_2 = \mathcal{B}_1 \cdot 2^{f-e}\mathbf{M}$

$R-P, S-P, R-Q, S-Q \leftarrow$ DiffPoints$(E, \mathcal{B}_1, \mathcal{B}_2)$
Normalize$(P, Q, P-Q, R, S, R-S, R-P, S-P, R-Q, S-Q, E)$
$\zeta \leftarrow$ Tate$(E, f, P, Q, P-Q)^{2^{f-e}}$
$\zeta_1 \leftarrow$ Tate$(E, e, R, Q, R-Q)$
$\zeta_2 \leftarrow$ Tate$(E, e, S, Q, S-Q)$
$\zeta_3 \leftarrow$ Tate$(E, e, R, P, R-P)$
$\zeta_4 \leftarrow$ Tate$(E, e, S, P, S-P)$
$\zeta_3 \leftarrow 1/\zeta_3$
$\zeta_4 \leftarrow 1/\zeta_4$
for i **from** 1 **up to** 4 **do**
  $x_i \leftarrow$ NormalizedDlog$(\zeta_i, 1/\zeta, e)$
return $\begin{pmatrix} x_1 & x_2 \\ x_3 & x_4 \end{pmatrix}$

Algorithm 4.107 BARYComponents — Cost: $6\mathbf{M} + 2\mathbf{S} + 9\mathbf{a}$

Input: P: ECPoint,
Q: ECPoint,
$P-Q$: ECPoint
Output: $(u : v : w)$: BaryCoords such that $P - Q = (u + v : w)$ and $P + Q = (u - v : w)$

$x_P, z_P \leftarrow P$
$x_Q, z_Q \leftarrow Q$
$x_{P-Q}, z_{P-Q} \leftarrow P - Q$
$t_0 \leftarrow x_P + z_P$
$t_1 \leftarrow x_P - z_P$
$t_2 \leftarrow x_Q + z_Q$
$t_3 \leftarrow x_Q - z_Q$
$t_0 \leftarrow t_0 \cdot t_3$
$t_1 \leftarrow t_1 \cdot t_2$
$t_2 \leftarrow t_1 + t_0$
$t_3 \leftarrow t_1 - t_0$
$t_0 \leftarrow t_2 \cdot z_{P-Q}$
$t_0 \leftarrow t_0^2$
$t_1 \leftarrow t_3 \cdot x_{P-Q}$
$t_2 \leftarrow t_3 \cdot z_{P-Q}$
$t_3 \leftarrow t_1^2$
$u \leftarrow t_3 + t_0$
$v \leftarrow t_3 - t_0$
$w \leftarrow t_1 \cdot t_2$
$w \leftarrow w + w$
return BaryCoords$(u : v : w)$

4.5. Higher dimensional isogenies (HD)

In the following, we detail the main algorithm Isogeny22Chain of the HD module to compute a $(2^e, 2^e)$-isogeny between products of elliptic curves.

The goal of this module is to compute isogenies in dimension 2. More precisely, the main function of this module Isogeny22Chain computes $(2^e, 2^e)$-isogenies between products of elliptic curves (with $e \leq f - 2$), as defined below.

Definition 4.5.1. Let $\Phi : E_1 \times E_2 \to E_3 \times E_4$ be an isogeny between elliptic products. It can be written as a matrix of elliptic curve isogenies

$$\Phi := \begin{pmatrix} \varphi_{11} & \varphi_{12} \\ \varphi_{21} & \varphi_{22} \end{pmatrix},$$

acting on points by matrix-vector multiplication on the left:

$$\Phi(P, Q) = (\varphi_{11}(P) + \varphi_{12}(Q), \varphi_{21}(P) + \varphi_{22}(Q)).$$

The *contravariant* $\widetilde{\Phi} : E_3 \times E_4 \to E_1 \times E_2$ of Φ is given by:

$$\widetilde{\Phi} := \begin{pmatrix} \widehat{\varphi}_{11} & \widehat{\varphi}_{21} \\ \widehat{\varphi}_{12} & \widehat{\varphi}_{22} \end{pmatrix}.$$

If $N \in \mathbb{Z}_{>0}$, we say that Φ is an (N, N)-*isogeny* if it satisfies $\widetilde{\Phi} \circ \Phi = [N]$, where $[N]$ is the multiplication by N in $E_1 \times E_2$.

If $\Phi\colon E_1 \times E_2 \to E_3 \times E_4$ is a $(2^e, 2^e)$-isogeny (with $e \leq f - 2$), Isogeny22Chain takes as input some representation of $\ker(\Phi)$ (up to additional data specified in Section 4.5) and a set $\mathcal{S}$ of points to evaluate and returns its codomain $E_3 \times E_4$ along with $\Phi(S)$ for all $S \in \mathcal{S}$.

4.5.1. Theta structures

On a product of elliptic curves $E_1 \times E_2$, points can be described as pairs (P, Q) of ECPoint. However, for intermediate computations in Isogeny22Chain (that will be presented in Section 4.5.4), we may need to represent points on 2-dimensional objects generalising these products, called *principally polarised abelian surfaces (PPAS)*. The reader only needs to know that PPAS are projective (*i.e.* admit projective systems of coordinates), have a commutative group law (like elliptic curves), and that the notion of (N, N)-isogeny introduced in Definition 4.5.1 generalises to PPAS.

Pairs of ECPoint are no longer suitable to represent points on general PPAS; we need a different coordinate system. We use (level-2) *theta coordinates*, introduced by Mumford [Mum66], as a system of coordinates on our PPAS. Concretely, given a PPAS A defined over $\mathbb{F}_{p^2}$, a level-2 *theta structure* (or system of theta coordinates) is an injective map $\theta^A\colon A(\mathbb{F}_{p^2}) \to \mathbb{P}^3(\mathbb{F}_{p^2}), P \mapsto \theta^A(P)$. This allows us to view A as a subvariety of projective space $\mathbb{P}^3(\mathbb{F}_{p^2})$, and work with points P on A as projective points $\theta^A(P) := (x_P : y_P : z_P : t_P)$.

Our choice to use the theta model as our coordinate system is motivated by the incredibly efficient and compact formulae it yields for performing arithmetic on A. This arises from a list of properties satisfied by the map θ^A. The key properties necessary for the reader to follow this section are the following (see [DMPR24, Section 2] for more details).

Proposition 4.5.2. *Let A be a PPAS.*

(1) If A is not a product[4] and $\theta^A\colon A(\mathbb{F}_{p^2}) \to \mathbb{P}^3(\mathbb{F}_{p^2})$ is a theta structure, then for all $P, Q \in A(\mathbb{F}_{p^2})$, $\theta^A(P) = \theta^A(Q)$ if and only if $P = \pm Q$. In other words, theta coordinates represent points up to sign.

(2) A non-product PPAS A with its theta structure (A, θ^A) can be represented by its theta null point *$\theta^A(0_A)$.*

(3) A PPAS A admits multiple theta structures. There is a surjective map:

$$\{\textit{Symplectic basis of } A[4]\} \longrightarrow \{\textit{Theta structures on } A\}$$

This means theta structures are determined by a symplectic basis of $A[4]$, as defined below.

Definition 4.5.3. Let A be a PPAS and $n \in \mathbb{Z}_{>0}$. Then it admits a n-th Weil pairing $w_n\colon A[n] \times A[n] \to \overline{\mathbb{F}}_p$. If $A = E_1 \times E_2$, then $w_n^A((P_1, P_2), (Q_1, Q_2)) = w_n^{E_1}(P_1, Q_1) w_n^{E_2}(P_2, Q_2)$, where the superscripts indicate the curves on which the Weil pairings are defined.

(1) We say that $P, Q \in A[n]$ *form an isotropic subgroup* if $w_n(P, Q) = 1$.

(2) A (ζ-)*symplectic basis* (S_1, S_2, T_1, T_2) of $A[n]$ is a basis of $A[n]$ such that:
- S_1, S_2 and T_1, T_2 and S_1, T_2 and S_2, T_1 form isotropic subgroups of $A[n]$;
- $w_n(S_1, T_1) = w_n(S_2, T_2) = \zeta$;

where $\zeta \in \overline{\mathbb{F}}_p$ is a primitive n-th root of unity.

4.5.1.1. Data types.

To work with theta structures and theta coordinates, we define the following types.

Definition 4.5.4. The type Dim2Structure represents a theta null point $\theta^A(0_A) = (a : b : c : d) \in \mathbb{P}^3(\mathbb{F}_{p^2})$ defining a PPAS A in the theta model. For simplicity, we will often denote this by 0_A unless it is necessary to remove ambiguity.

Definition 4.5.5. The type Dim2Point represents a point $\theta^A(P) = (x : y : z : t) \in \mathbb{P}^3(\mathbb{F}_{p^2})$ on a PPAS A in theta coordinates. For simplicity, we will often denote this simply by P unless it is necessary to remove ambiguity. The function Dim2Coordinates takes a point P and returns its coordinates x, y, z, t as a list.

To represent points on product PPAS, we define the following type.

Definition 4.5.6. The type ECPointPair represents a couple (P, Q) of ECPoint on a product of Montgomery curves.

[4] Which means that A is not isomorphic (as a PPAS) to a product of elliptic curves.

4.5.2. Operations on points

4.5.2.1. Coordinate-wise operations. Let θ^A be a theta structure on a PPAS A and $P, Q \in A$. There are three key coordinate-wise operations we perform on four points:

- $\theta^A(P) \odot \theta^A(Q) := (x_P \cdot x_Q : y_P \cdot y_Q : z_P \cdot z_Q : t_P \cdot t_Q)$ the coordinate-wise product, that can be computed with PointwiseMult.
- $\theta^A(P)^{\odot 2} := \theta^A(P) \odot \theta^A(P)$ the squaring, that can be computed via Squaring.
- $\theta^A(P)^{\odot -1} := (x_P^{-1} : y_P^{-1} : z_P^{-1} : t_P^{-1})$ the projective inverse, that can be computed via Inverse without inversion.
- $H(\theta^A(P))$ the Hadamard transform of $\theta^A(P)$, that can be computed with Hadamard. Note that H acts on Dim2Point by multiplication on the left by the matrix:

$$\begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{pmatrix}.$$

Algorithm 4.108 PointwiseMult — Cost: 4**M**

Input: P: Dim2Point
Q: Dim2Point
Output: R: Dim2Point

1: $x_P, y_P, z_P, t_P \leftarrow$ Dim2Coordinates(P)
2: $x_Q, y_Q, z_Q, t_Q \leftarrow$ Dim2Coordinates(Q)
3: $X \leftarrow x_P \cdot x_Q$
4: $Y \leftarrow y_P \cdot y_Q$
5: $Z \leftarrow z_P \cdot z_Q$
6: $T \leftarrow t_P \cdot t_Q$
7: **return** Dim2Point $((X : Y : Z : T))$

Algorithm 4.109 Squaring — Cost: 4**S**

Input: P: Dim2Point
Output: Q: Dim2Point // $Q = P^{\odot 2}$

1: $x, y, z, t \leftarrow$ Dim2Coordinates(P)
2: $X \leftarrow x \cdot x$
3: $Y \leftarrow y \cdot y$
4: $Z \leftarrow z \cdot z$
5: $T \leftarrow t \cdot t$
6: **return** Dim2Point $((X : Y : Z : T))$

Algorithm 4.110 Inverse — Cost: 6**M**

Input: P: Dim2Point
Output: Q: Dim2Point // $Q = P^{\odot -1}$

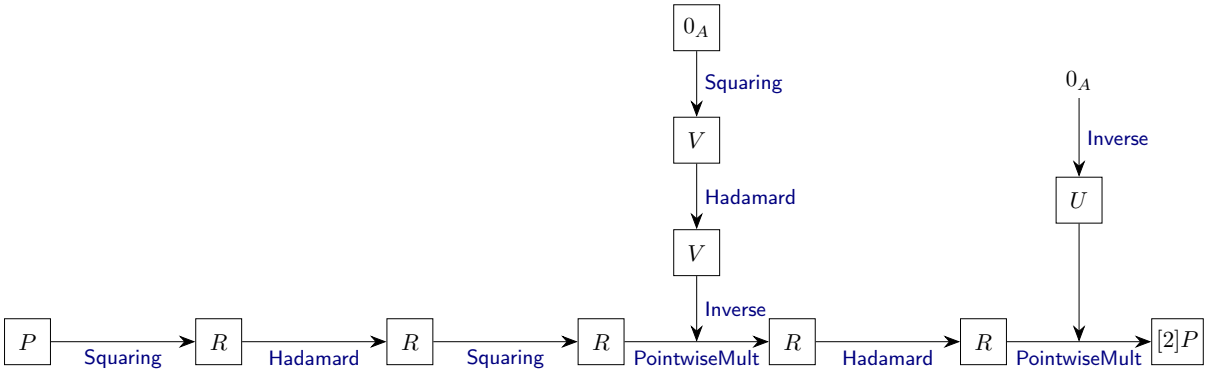
1: $x, y, z, t \leftarrow$ Dim2Coordinates(P)
2: $t_1 \leftarrow x \cdot y$
3: $t_2 \leftarrow z \cdot t$
4: $X \leftarrow t_2 \cdot y$
5: $Y \leftarrow t_2 \cdot x$
6: $Z \leftarrow t_1 \cdot t$
7: $T \leftarrow t_1 \cdot z$
8: **return** Dim2Point $((X : Y : Z : T))$

Algorithm 4.111 Hadamard — Cost: 8**a**

Input: P: Dim2Point
Output: Q: Dim2Point

$x, y, z, t \leftarrow$ Dim2Coordinates(P)
$t_1 \leftarrow x + y$
$t_2 \leftarrow x - y$
$t_3 \leftarrow z + t$
$t_4 \leftarrow z - t$
$X \leftarrow t_1 + t_3$
$Y \leftarrow t_2 + t_4$
$Z \leftarrow t_1 - t_3$
$T \leftarrow t_2 - t_4$
return Dim2Point $((X : Y : Z : T))$

4.5.2.2. Doubling. Let P be a Dim2Point on a PPAS A. A key operation in the computation of $(2, 2)$-isogenies is point doubling, i.e. computing $[2]P$. The doubling procedure is summarised in the following diagram.



The constants $U = \theta^A(0_A)^{\odot -1}$ and $V = H(\theta^A(0_A)^{\odot 2})^{\odot -1}$ depend only on the PPAS A and thus can be precomputed from $\theta^A(0_A)$ with ComputeConstants. With these constants and P as input, Dim2DBL uses the procedure above to return $[2]P$, a Dim2Point on A. For optimisation purposes in isogeny computations, we may not be given the theta null-point $\theta^A(0_A)$ to represent a theta structure but its inverse $\theta^A(0_A)^{\odot -1}$ instead along with auxiliary data. ComputeConstantsFromIsoData computes the necessary constants U and V in that case.

Algorithm 4.112 ComputeConstants — Cost: $12\mathbf{M} + 4\mathbf{S} + 8\mathbf{a}$

Input: A: Dim2Structure
Output: U: Dim2Point // $U = \theta^A(0_A)^{\odot -1}$
V: Dim2Point // $V = H(\theta^A(0_A)^{\odot 2})^{\odot -1}$

$0_A \leftarrow A$
$U \leftarrow$ Inverse(0_A)
$V \leftarrow$ Squaring(0_A)
$V \leftarrow$ Hadamard(V)
$V \leftarrow$ Inverse(V)
return U, V

4.5.3. The dual structure

We note that H has order 2 as a linear map: for $P \in \mathbb{P}^3(\mathbb{F}_{p^2})$ we have $H(H(P)) = P$. The Hadamard therefore allows us to define a dual theta structure on a PPAS A from a theta structure θ^A on A:

Definition 4.5.7. If θ^A is a theta structure on a PPAS A, then $\widetilde{\theta}^A := H \circ \theta^A$ defines a theta structure on A called the *dual theta structure* of θ^A.

Algorithm 4.113 ComputeConstantsFromIsoData — Cost: $9\mathbf{M} + 4\mathbf{S} + 8\mathbf{a}$

Input: U: Dim2Point — // $U = \theta^A(0_A)^{\odot -1}$
x_1, x_2, y_1, y_2: Fp2Elem — // Auxiliary constants outputed by GenericCodomain
Output: A: Dim2Structure — // Theta structure of theta null-point $\theta^A(0_A)$
V: Dim2Point — // $V = H(\theta^A(0_A)^{\odot 2})^{\odot -1}$

1: $a^{-1}, b^{-1}, c^{-1}, d^{-1} \leftarrow$ Dim2Coordinates(U)
2: $u \leftarrow x_2 \cdot y_2$
3: $a' \leftarrow u \cdot x_1$
4: $b' \leftarrow u \cdot y_1$
5: $c' \leftarrow d^{-1}$
6: $d' \leftarrow c^{-1}$
7: $0_A \leftarrow$ Dim2Point$((a' : b' : c' : d'))$
8: $A \leftarrow$ Dim2Structure(0_A)
9: $V \leftarrow$ Squaring(0_A)
10: $V \leftarrow$ Hadamard(V)
11: $V \leftarrow$ Inverse(V)
12: **return** A, V

Algorithm 4.114 Dim2DBL — Cost: $8\mathbf{M} + 8\mathbf{S} + 16\mathbf{a}$

Input: P: Dim2Point
U: Dim2Point — // Output of ComputeConstants $U = \theta^A(0_A)^{\odot -1}$
V: Dim2Point — // Output of ComputeConstants $V = H(\theta^A(0_A)^{\odot 2})^{\odot -1}$
Output: R: Dim2Point

1: $R \leftarrow$ Squaring(P)
2: $R \leftarrow$ Hadamard(R)
3: $R \leftarrow$ Squaring(R)
4: $R \leftarrow$ PointwiseMult(R, V)
5: $R \leftarrow$ Hadamard(R)
6: $R \leftarrow$ PointwiseMult(R, U)
7: **return** R

From the discussion above, we see that $\theta^A = H \circ \widetilde{\theta}^A$. Certain computations will be more efficient on the dual theta structure, and the Hadamard map allows us to switch between the two efficiently.

4.5.4. Algorithmic overview

Let $\Phi\colon E_1 \times E_2 \to E_3 \times E_4$ be a $(2^e, 2^e)$-isogeny (with $e \leq f - 2$) to compute with Isogeny22Chain. In our applications, Φ will always be derived from an application of Kani's lemma [Kan97] (see also [MMP+23, Theorem 1]), *i.e.*, it will have a matrix expression of the form:

$$\Phi := \begin{pmatrix} \varphi_1 & \widehat{\varphi}_2 \\ -\psi_2 & \widehat{\psi}_1 \end{pmatrix},$$

so that it acts on points by matrix-vector multiplication on the left as follows:

$$\Phi(P, Q) = (\varphi_1(P) + \widehat{\varphi}_2(Q), -\psi_2(P) + \widehat{\psi}_1(Q)),$$

where $\varphi_1, \varphi_2, \psi_1, \psi_2$ fit into the commutative diagram:

$$\begin{array}{ccc} E_4 & \xrightarrow{\psi_1} & E_2 \\ {\scriptstyle \psi_2}\uparrow & \circlearrowleft & \uparrow{\scriptstyle \varphi_2} \\ E_1 & \xrightarrow{\varphi_1} & E_3 \end{array}$$

with $\deg(\varphi_1) = \deg(\psi_1) := N_1$, $\deg(\varphi_2) = \deg(\psi_2) := N_2$, $\gcd(N_1, N_2) = 1$ and $N_1 + N_2 = 2^e$.

4.5.4.1. Kernel inputs. The isogeny Φ can always be computed from its kernel:

$$\ker(\Phi) = \{([N_1]P, \varphi_2 \circ \varphi_1(P)) \mid P \in E_1[2^e]\} \subseteq (E_1 \times E_2)[2^e],$$

but this computation involves square roots [DMPR24, Section 4.2]. To remove the need for square roots, we instead use 2^{e+2}-torsion points to construct our kernel. Note that the constraint $e \leq f-2$ allows us to have access to $\mathbb{F}_{p^2}$-rational 2^{e+2}-torsion points. Isogeny22Chain is given as input two points $T_1, T_2 \in (E_1 \times E_2)[2^{e+2}]$ such that $\ker(\Phi) = \langle [4]T_1, [4]T_2 \rangle$ and that form an *isotropic subgroup*, *i.e.*, with trivial Weil pairing $w_{2^{e+2}}(T_1, T_2) = 1$.

4.5.4.2. Computational steps. As its name indicates, Isogeny22Chain divides the computation of Φ into a chain of length e of $(2,2)$-isogenies between PPAS:

$$E_1 \times E_2 \xrightarrow[\text{Gluing}]{\Phi_1} A_1 \xrightarrow{\Phi_2} A_2 \quad \cdots \quad A_{e-2} \xrightarrow{\Phi_{e-1}} A_{e-1} \xrightarrow[\text{Splitting}]{\Phi_e} A_e = E_3 \times E_4$$
$$\text{Generic}$$

so that $\Phi = \Phi_e \circ \cdots \circ \Phi_1$. It uses level-2 theta coordinates to compute each $(2,2)$-isogeny of the chain, following the algorithmic approach of [DMPR24]. We assume that for all $i \in \{1, \cdots, e-1\}$, A_i is a PPAS which is not a product of elliptic curves. Indeed, heuristically, the probability that A_i is a product is expected to be negligible $O(p^{-1})$. Therefore, $\Phi_1 \colon E_1 \times E_2 \to A_1$ is a *gluing isogeny* and $\Phi_e \colon A_{e-1} \to E_3 \times E_4$ is a *splitting isogeny*.

The computation of Φ is then naturally divided into three phases:

(1) Computing the codomain theta null point 0_{A_1} of the gluing isogeny from the 8-torsion points $[2^{e-1}]T_1$ and $[2^{e-1}]T_2$ lying above its kernel. This is the most technical step of the isogeny chain, and special evaluation functions need to be used; these specialised algorithms are detailed in Section 4.5.6.
(2) For all $i \in \{2, \cdots, e\}$, compute the theta null point 0_{A_i} of the codomain of $\Phi_i \colon A_{i-1} \to A_i$ from the 8-torsion points $[2^{e-i}]\Phi_{i-1} \circ \cdots \circ \Phi_1(T_1)$ and $[2^{e-i}]\Phi_{i-1} \circ \cdots \circ \Phi_1(T_2)$ lying above its kernel. The computation also holds for the splitting isogeny ($i = e$) and is detailed in Section 4.5.5.
(3) From the last codomain theta null point 0_{A_e}, we split A_e into an elliptic product $E_3 \times E_4$ to extract the equations of E_3 and E_4, and coordinates of image points. This is detailed in Section 4.5.7.

4.5.5. Generic $(2,2)$-isogenies

We first explain the computation of a generic $(2,2)$-isogeny $\Phi_i \colon A_{i-1} \to A_i$ ($i \geq 2$), the most common operation in a $(2,2)$-isogeny chain computation. As there are multiple choices of theta structures that can be put on A_{i-1} and A_i, we need to fix these choices to ensure they are compatible for the efficient isogeny computation to work. We ensure that these conditions are satisfied throughout the generic isogeny computation phase of the chain.

Assume that we want to compute a $(2,2)$-isogeny $\varphi \colon A \to B$ between non-product PPAS, and that we are given theta coordinates of 8-torsion points $K_1, K_2 \in A[8]$ such that $\ker(\varphi) = \langle [4]K_1, [4]K_2 \rangle$. We further assume that θ^A is a theta structure on A that is *adapted* to φ in the sense of the following definition (Definition 4.5.8).

Definition 4.5.8. A theta structure θ^A on A is *adapted* to φ with respect to a symplectic basis $\mathscr{B} := (J_1, J_2, K_1, K_2)$ of $A[8]$ if:

(1) $\ker(\Phi) = \langle [4]K_1, [4]K_2 \rangle$; and
(2) $[2]\mathscr{B}$ induces θ^A via Proposition 4.5.2.(3).

With these inputs, GenericCodomain computes the inverse of the codomain dual theta null-point $\widetilde{\theta}^B(0_B)^{\odot -1}$ on B. To ensure that the input is valid, namely that the input kernel is valid (in the sense above), we use VerifyKernel. This procedure is only necessary in the verification of the signature, thus we use a boolean `verify_kernel` to indicate if it is run. We then use the codomain dual theta null-point to evaluate points, as shown in GenericEval. Correctness of both algorithms follows from [DMPR24]; more details can also be found in Appendix A.1.1.

Remark 4.5.9. For optimisation purposes, GenericCodomain and GenericEval can take as inputs points expressed with the theta structure θ^A or its dual $\widetilde{\theta}^A$. The input variable `hadamard` should specify this choice for both functions, where a `true` boolean value means that we use dual coordinates.

Remark 4.5.10. To avoid redundant computations, constant precomputations for point doublings are not performed on every intermediate PPAS, but only on PPAS where point doublings are computed. On these PPAS, ComputeConstantsFromIsoData uses the outputs x_1, y_1, x_2, y_2 from GenericCodomain to compute points $U = \widetilde{\theta}^B(0_B)^{\odot -1}$ and $V = H(\widetilde{\theta}^B(0_B)^{\odot 2})^{\odot -1}$, which are then used to perform point doublings on B with Dim2DBL.

Algorithm 4.115 VerifyKernel — Cost: 2M

Input: P: Dim2Point // $P = H(\theta^A(K_1)^{\odot 2})$
U: Dim2Point // $U = \widetilde{\theta^B}(0_B)^{\odot -1}$
Output: `bool` : boolean

$x_P, y_P, z_P, t_P \leftarrow$ Dim2Coordinates(P)
$x_U, y_U, z_U, t_U \leftarrow$ Dim2Coordinates(U)
$t_1 \leftarrow z_P \cdot z_U$
$t_2 \leftarrow t_P \cdot t_U$
if not $t_1 == t_2$ **then**
    return `false`
return `true`

Algorithm 4.116 GenericCodomain — Cost: $6\mathbf{M} + 8\mathbf{S} + 16\mathbf{a} + 2\mathbf{M} \cdot$ `verify_kernel` $+ 16\mathbf{a} \cdot$ `hadamard`

Input: A: Dim2Structure
K_1: Dim2Point // An 8-torsion point
K_2: Dim2Point // An 8-torsion point
`verify_kernel` : boolean
`hadamard` : boolean
Output: U: Dim2Point // $U = \widetilde{\theta^B}(0_B)^{\odot -1}$
x_1, x_2, y_1, y_2: Fp2Elem // ComputeConstantsFromIsoData inputs $(x_i : y_i : \cdots) = H(\theta^A(K_i)^{\odot 2})$

if `hadamard` **then**
    $P_1 \leftarrow$ Hadamard(K_1)
    $P_2 \leftarrow$ Hadamard(K_2)
else
    $P_1 \leftarrow K_1$
    $P_2 \leftarrow K_2$
$P_1 \leftarrow$ Squaring(P_1)
$P_2 \leftarrow$ Squaring(P_2)
$P_1 \leftarrow$ Hadamard(P_1)
$P_2 \leftarrow$ Hadamard(P_2)
$x_1, y_1, z_1, t_1 \leftarrow$ Dim2Coordinates(P_1)
$x_2, y_2, z_2, t_2 \leftarrow$ Dim2Coordinates(P_2)
if $0 \in \{x_2, y_2, z_2, t_2, x_1, y_1\}$ **then**
    raise Exception ("GenericCodomain failed: unexpected splitting occurred")
$u_1 \leftarrow y_1 \cdot t_2$
$u_2 \leftarrow x_1 \cdot z_2$
$A^{-1} \leftarrow u_1 \cdot z_2$
$B^{-1} \leftarrow u_2 \cdot t_2$
$C^{-1} \leftarrow u_1 \cdot x_2$
$D^{-1} \leftarrow u_2 \cdot y_2$
$U \leftarrow$ Dim2Point$(A^{-1} : B^{-1} : C^{-1} : D^{-1})$
if `verify_kernel` **then**
    if not VerifyKernel(P_1, U) **then**
        raise Exception ("GenericCodomain failed: The kernel generated by K_1, K_2 is not valid")
return U, x_1, x_2, y_1, y_2

4.5.6. The gluing $(2,2)$-isogeny

The gluing $\Phi_1 \colon E_1 \times E_2 \to A_1$ is the first step of the isogeny chain computation. The following algorithms are due to [Dup25]. To proceed, we need to find a theta structure θ^{A_0} on the domain $A_0 := E_1 \times E_2$ that we do not know *a priori* because we are only given ECPoint data in $(X : Z)$-coordinates. This theta structure θ^{A_0} should be adapted to Φ_1 in the sense of Definition 4.5.8.

Actually, by choosing a symplectic basis $\mathscr{B} := (S_1, S_2, T_1, T_2)$ of $A_0[2^{e+2}]$ adapted to Φ, we can not only ensure that θ^{A_0} is adapted to Φ_1 but also that $\theta^{A_{i-1}}$ is adapted to Φ_i and compatible with θ^{A_i} for all $1 \leq i \leq e$, as required to apply GenericCodomain at every generic step $i \geq 2$ (see Proposition A.1.1).

The gluing procedure proceeds as follows.

(1) Let T_1 and T_2 be the input kernel points. We can assume that they are given in a specific form, which allows us to choose S_1 and S_2 completing a symplectic basis $\mathscr{B} := (S_1, S_2, T_1, T_2)$ of $A_0[2^{e+2}]$.
(2) Using this data, from the $(X : Z)$-coordinates of E_1 and E_2, we can then compute the theta structure θ^{A_0}. This is explained in Section 4.5.6.1.
(3) We then use GluingCodomain to determine the theta structure θ^{A_1}. This is explained in Section 4.5.6.2.

Algorithm 4.117 GenericEval — Cost: $4\mathbf{M} + 4\mathbf{S} + 8\mathbf{a} + 8\mathbf{a} \cdot$ `hadamard`

Input: `pts` : array of Dim2Point
U : Dim2Point // Precomputed point output by codomain functions $U = \widetilde{\theta}^B(0_B)^{\odot -1}$
`hadamard` : boolean // Should match that used in codomain function
Output: `image_pts` : array of Dim2Point

```
Initialise empty array image_pts
for each R in pts do
    if hadamard then
        P ← Hadamard(R)
    else
        P ← R
    P ← Squaring(P)
    P ← Hadamard(P)
    P ← PointwiseMult(P, U)
    Append P to image_pts
return image_pts
```

To proceed with the generic isogeny computations, we first need to evaluate Φ_1 given some data that represents θ^{A_1}: as the dual codomain theta null-point has a zero last coordinate, we must also use the images under Φ_1 of the 8-torsion points $[2^{e-1}]T_1, [2^{e-1}]T_2$. This information is therefore also output by GluingCodomain. We can then evaluate Φ_1 using GluingEval, which is explained in Section 4.5.6.3.

4.5.6.1. Computing a theta structure adapted to a gluing $(2,2)$-isogeny. To simplify the notation in the following, let $\varphi\colon A := E_1 \times E_2 \to B$ denote a gluing $(2,2)$-isogeny that we want to compute. Assume that B is a non-product PPAS and that we are given 8-torsion points $K_1 := (K_{1,1}, K_{1,2}), K_2 := (K_{2,1}, K_{2,2}) \in A[8]$ such that $\ker(\varphi) = \langle [4]K_1, [4]K_2 \rangle$. Since φ is a gluing isogeny, $J_1 := (0, K_{2,2})$ and $J_2 := (K_{1,1}, 0)$ have order 8 and $\mathscr{B}_0 := (J_1, J_2, K_1, K_2)$ is a symplectic basis of $A[8]$ by [Dar25, Lemma 6.5.3].

Given the elements of $[2]\mathscr{B}_0$ as inputs, ThetaChangeOfCoordinates outputs the *change of coordinates matrix* $\mathbf{M} \in M_4(\mathbb{F}_{p^2})$ such that, given $(x_1 : z_1) \in E_1$ and $(x_2 : z_2) \in E_2$, we compute the corresponding point $P \in A$ as

$$\theta^A(P) = (x : y : z : t)^T = \mathbf{M}(x_1x_2 : x_1z_2 : z_1x_2 : z_1z_2)^T.$$

In this way, we compute the theta structure θ^A induced by $[2]\mathscr{B}_0$ (adapted to φ). This ensures that $\theta^{A_{i-1}}$ is adapted to Φ_i for all $i \geq 2$ in subsequent applications of GenericCodomain (and thus VerifyKernel).

As shown by [Dup25, Section 3.2], to compute $\mathbf{M}$, for some $P \in E[4]$, we need to compute the (2×2) matrix $\mathfrak{g}_P$ giving addition by $[2]P$ on E in Montgomery coordinates. This is done by CoSymElem.[5]

4.5.6.2. Computing the codomain of a gluing $(2,2)$-isogeny. The dual codomain theta null-point has a zero last coordinate $\widetilde{\theta}^B(0_B) = (\alpha : \beta : \gamma : 0)$ by [Dar25, Lemma 6.5.14]. So, its inverse is not defined and may not be used to evaluate any point with Equation (22). As a consequence, GluingCodomain not only computes the constants α, β, γ and their projective inverse, but also the projective inverse of $\widetilde{\theta}^B(\varphi(K_1))$ and $\widetilde{\theta}^B(\varphi(K_2))$, which are not expected to vanish.

4.5.6.3. Evaluating a gluing $(2,2)$-isogeny. Instead of using (the simpler) Equation (22) as in the generic case, we generally use the following equation to evaluate a point $P \in A = E_1 \times E_2$:

$$\theta^B(\varphi(P)) = H\left(H(\theta^A(P - W) \odot \theta^A(P + W)) \odot \widetilde{\theta}^B(W)^{\odot -1}\right), \tag{9}$$

where $W \in A$ is a point of known and invertible image $\widetilde{\theta}^B(\varphi(W))$ (*e.g.*, K_1 or K_2). Using [Dup25, Lemma 2], one can compute the values of $\theta^A(P - W)$ and $\theta^A(P + W)$ simultaneously and efficiently from the barycentric coordinates (BaryCoords) of the P_i and W_i ($i \in \{1, 2\}$), as defined in Definition 4.4.3. These BaryCoords elements should be given as input of GluingEval, and may be computed from the P_i, W_i and $P_i - W_i$ with BARYComponents.

[5] We follow the terminology of [Dup25] and call the elements $\mathfrak{g}_P$ *symmetric elements*, giving rise to the algorithm CoSymElem.

Algorithm 4.118 CoSymElem — Cost: $3\mathbf{M} + 2\mathbf{S} + 7\mathbf{a} + 8\mathbf{n} + 3\mathbf{i}$

Input: P: ECPoint
Q: ECPoint // (P, Q) is a basis of the 4-torsion
Output: $\mathfrak{g}_P$: 2×2 matrix of Fp2Elem // Associated to P
$\mathfrak{g}_Q$: 2×2 matrix of Fp2Elem // Associated to Q
$\mathfrak{g}_{P+Q}$: 2×2 matrix of Fp2Elem // Associated to $P+Q$
λ: Fp2Elem // Projective factor

$x_P, z_P \leftarrow P$; $x_Q, z_Q \leftarrow Q$
$s_P \leftarrow x_P + z_P$; $s_Q \leftarrow x_Q + z_Q$; $t_P \leftarrow x_P - z_P$; $t_Q \leftarrow x_Q - z_Q$
$t_P \leftarrow t_P \cdot s_Q$; $t_Q \leftarrow t_Q \cdot s_P$; $t_Q \leftarrow \mathbf{i} \cdot t_Q$
if $x_P == z_P$ **then**
  $x_2 \leftarrow x_Q^2$; $z_2 \leftarrow z_Q^2$; $\lambda \leftarrow x_2 - z_2$
  $s_2 \leftarrow x_2 + z_2$; $m_2 \leftarrow x_Q \cdot z_Q$; $m_2 \leftarrow m_2 + m_2$; $t_2 \leftarrow \mathbf{i} \cdot s_2$; $n_2 \leftarrow \mathbf{i} \cdot m_2$
  $\mathfrak{g}_P \leftarrow \begin{pmatrix} 0 & \lambda \\ \lambda & 0 \end{pmatrix}$; $\mathfrak{g}_Q \leftarrow \begin{pmatrix} s_2 & -m_2 \\ m_2 & -s_2 \end{pmatrix}$; $\mathfrak{g}_{P+Q} \leftarrow \begin{pmatrix} t_2 & -n_2 \\ n_2 & -t_2 \end{pmatrix}$
else if $x_P == -z_P$ **then**
  $x_2 \leftarrow x_Q^2$; $z_2 \leftarrow z_Q^2$; $\lambda \leftarrow x_2 - z_2$
  $s_2 \leftarrow x_2 + z_2$; $m_2 \leftarrow x_Q \cdot z_Q$; $m_2 \leftarrow m_2 + m_2$; $t_2 \leftarrow \mathbf{i} \cdot s_2$; $n_2 \leftarrow \mathbf{i} \cdot m_2$
  $\mathfrak{g}_P \leftarrow \begin{pmatrix} 0 & -\lambda \\ -\lambda & 0 \end{pmatrix}$; $\mathfrak{g}_Q \leftarrow \begin{pmatrix} s_2 & -m_2 \\ m_2 & -s_2 \end{pmatrix}$; $\mathfrak{g}_{P+Q} \leftarrow \begin{pmatrix} -t_2 & n_2 \\ -n_2 & t_2 \end{pmatrix}$
else if $x_Q == z_Q$ **then**
  $x_2 \leftarrow x_P^2$; $z_2 \leftarrow z_P^2$; $\lambda \leftarrow x_2 - z_2$
  $s_2 \leftarrow x_2 + z_2$; $m_2 \leftarrow x_P \cdot z_P$; $m_2 \leftarrow m_2 + m_2$; $t_2 \leftarrow \mathbf{i} \cdot s_2$; $n_2 \leftarrow \mathbf{i} \cdot m_2$
  $\mathfrak{g}_P \leftarrow \begin{pmatrix} s_2 & -m_2 \\ m_2 & -s_2 \end{pmatrix}$; $\mathfrak{g}_Q \leftarrow \begin{pmatrix} 0 & \lambda \\ \lambda & 0 \end{pmatrix}$; $\mathfrak{g}_{P+Q} \leftarrow \begin{pmatrix} -t_2 & n_2 \\ -n_2 & t_2 \end{pmatrix}$
else if $x_Q == -z_Q$ **then**
  $x_2 \leftarrow x_P^2$; $z_2 \leftarrow z_P^2$; $\lambda \leftarrow x_2 - z_2$
  $s_2 \leftarrow x_2 + z_2$; $m_2 \leftarrow x_P \cdot z_P$; $m_2 \leftarrow m_2 + m_2$; $t_2 \leftarrow \mathbf{i} \cdot s_2$; $n_2 \leftarrow \mathbf{i} \cdot m_2$
  $\mathfrak{g}_P \leftarrow \begin{pmatrix} s_2 & -m_2 \\ m_2 & -s_2 \end{pmatrix}$; $\mathfrak{g}_Q \leftarrow \begin{pmatrix} 0 & -\lambda \\ -\lambda & 0 \end{pmatrix}$; $\mathfrak{g}_{P+Q} \leftarrow \begin{pmatrix} t_2 & -n_2 \\ n_2 & -t_2 \end{pmatrix}$
else if $t_Q == -t_P$ **then**
  $x_2 \leftarrow x_P^2$; $z_2 \leftarrow z_P^2$; $\lambda \leftarrow x_2 - z_2$
  $s_2 \leftarrow x_2 + z_2$; $m_2 \leftarrow x_P \cdot z_P$; $m_2 \leftarrow m_2 + m_2$; $t_2 \leftarrow \mathbf{i} \cdot s_2$; $n_2 \leftarrow \mathbf{i} \cdot m_2$
  $\mathfrak{g}_P \leftarrow \begin{pmatrix} s_2 & -m_2 \\ m_2 & -s_2 \end{pmatrix}$; $\mathfrak{g}_Q \leftarrow \begin{pmatrix} -t_2 & n_2 \\ -n_2 & t_2 \end{pmatrix}$; $\mathfrak{g}_{P+Q} \leftarrow \begin{pmatrix} 0 & -\lambda \\ -\lambda & 0 \end{pmatrix}$
else if $t_Q == t_P$ **then**
  $x_2 \leftarrow x_P^2$; $z_2 \leftarrow z_P^2$; $\lambda \leftarrow x_2 - z_2$
  $s_2 \leftarrow x_2 + z_2$; $m_2 \leftarrow x_P \cdot z_P$; $m_2 \leftarrow m_2 + m_2$; $t_2 \leftarrow \mathbf{i} \cdot s_2$; $n_2 \leftarrow \mathbf{i} \cdot m_2$
  $\mathfrak{g}_P \leftarrow \begin{pmatrix} s_2 & -m_2 \\ m_2 & -s_2 \end{pmatrix}$; $\mathfrak{g}_Q \leftarrow \begin{pmatrix} t_2 & -n_2 \\ n_2 & -t_2 \end{pmatrix}$; $\mathfrak{g}_{P+Q} \leftarrow \begin{pmatrix} 0 & \lambda \\ \lambda & 0 \end{pmatrix}$
else
  raise Exception ("CoSymElem failed: input points do not form a basis of the 4-torsion of a Montgomery curve")
return $\mathfrak{g}_P, \mathfrak{g}_Q, \mathfrak{g}_{P+Q}, \lambda$

Remark 4.5.11. In Isogeny22Chain, the points P we shall need to evaluate will be multiples of input points T_1 or $T_2 \in (E_1 \times E_2)[2^{e+2}]$, where $K_1 = [2^{e-1}]T_1$ and $K_2 = [2^{e-1}]T_2$. To obtain the $P_i - W_i$ easily from either T_1 or T_2, we shall then set $W = T_1$ or T_2, depending on the value of P in order to use a Montgomery ladder. This is the reason why we make the effort of computing both $\widetilde{\theta}^B(\varphi(K_1))^{\odot -1}$ and $\widetilde{\theta}^B(\varphi(K_2))^{\odot -1}$ in GluingCodomain.

There is a special case when Equation (22) may still be applied to evaluate a point, which makes use of the inverse dual theta null point $\widetilde{\theta}^B(0_B)^{\odot -1} = (\alpha^{-1} : \beta^{-1} : \gamma^{-1} : 0)$. Namely, when P is of the form $P = (P_1, 0)$ or $(0, P_2)$. The algorithm GluingEvalSpecial treats this special case at much lower cost than GluingCodomain. Note that all points of $\mathcal{S}$ to evaluate provided as input to Isogeny22Chain are expected to be of that special form, so GluingEvalSpecial is indeed useful. For a proof of this, we refer to Appendix A.1.2.2.

Algorithm 4.119 ThetaChangeOfCoordinates — Cost: $55\mathbf{M} + 4\mathbf{S} + 35\mathbf{a} + 16\mathbf{n} + 6\mathbf{i}$

Input: K'_1: ECPointPair
K'_2: ECPointPair // $\ker(\varphi) = \langle [2]K'_1, [2]K'_2 \rangle$
Output: $\mathbf{M}$: 4×4 matrix of Fp2Elem // Change of coordinates matrix

$K'_{1,1}, K'_{1,2} \leftarrow K'_1$
$K'_{2,1}, K'_{2,2} \leftarrow K'_2$
$\mathbf{G}^1, \mathbf{G}^2, \mathbf{G}^3, \lambda_1 \leftarrow \text{CoSymElem}(K'_{1,1}, K'_{2,1})$
$\mathbf{H}^1, \mathbf{H}^2, \mathbf{H}^3, \lambda_2 \leftarrow \text{CoSymElem}(K'_{1,2}, K'_{2,2})$
$\lambda_{12} \leftarrow \lambda_1 \cdot \lambda_2$
$M_{0,0} \leftarrow \lambda_{12} + G^1_{0,0} \cdot H^1_{0,0} + G^2_{0,0} \cdot H^2_{0,0} - G^3_{0,0} \cdot H^3_{0,0}$
$M_{0,1} \leftarrow G^1_{0,0} \cdot H^1_{0,1} + G^2_{0,0} \cdot H^2_{0,1} - G^3_{0,0} \cdot H^3_{0,1}$
$M_{0,2} \leftarrow G^1_{0,1} \cdot H^1_{0,0} + G^2_{0,1} \cdot H^2_{0,0} - G^3_{0,1} \cdot H^3_{0,0}$
$M_{0,3} \leftarrow G^1_{0,1} \cdot H^1_{0,1} + G^2_{0,1} \cdot H^2_{0,1} - G^3_{0,1} \cdot H^3_{0,1}$
$M_{1,0} \leftarrow M_{0,0} \cdot H^2_{0,0} + M_{0,1} \cdot H^2_{1,0}$
$M_{1,1} \leftarrow M_{0,0} \cdot H^2_{0,1} - M_{0,1} \cdot H^2_{0,0}$
$M_{1,2} \leftarrow M_{0,2} \cdot H^2_{0,0} + M_{0,3} \cdot H^2_{1,0}$
$M_{1,3} \leftarrow M_{0,2} \cdot H^2_{0,1} - M_{0,3} \cdot H^2_{0,0}$
$M_{2,0} \leftarrow M_{0,0} \cdot G^2_{0,0} + M_{0,2} \cdot G^2_{1,0}$
$M_{2,1} \leftarrow M_{0,1} \cdot G^2_{0,0} + M_{0,3} \cdot G^2_{1,0}$
$M_{2,2} \leftarrow M_{0,0} \cdot G^2_{0,1} - M_{0,2} \cdot G^2_{0,0}$
$M_{2,3} \leftarrow M_{0,1} \cdot G^2_{0,1} - M_{0,3} \cdot G^2_{0,0}$
$M_{3,0} \leftarrow M_{1,0} \cdot G^2_{0,0} + M_{1,2} \cdot G^2_{1,0}$
$M_{3,1} \leftarrow M_{1,1} \cdot G^2_{0,0} + M_{1,3} \cdot G^2_{1,0}$
$M_{3,2} \leftarrow M_{1,0} \cdot G^2_{0,1} - M_{1,2} \cdot G^2_{0,0}$
$M_{3,3} \leftarrow M_{1,1} \cdot G^2_{0,1} - M_{1,3} \cdot G^2_{0,0}$
for j **from** 0 **up to** 3 **do**
    $M_{0,j} \leftarrow \lambda_{12} \cdot M_{0,j}$
    $M_{1,j} \leftarrow \lambda_1 \cdot M_{1,j}$
    $M_{2,j} \leftarrow \lambda_2 \cdot M_{2,j}$
$\mathbf{M} \leftarrow (M_{i,j})_{0 \le i,j \le 3}$
return $\mathbf{M}$

Algorithm 4.120 ApplyMatrix — Cost: $12\mathbf{M} + 8\mathbf{a} + (4\mathbf{M} + 4\mathbf{a})$

Input: $\mathbf{M}$: 4×4 matrix of Fp2Elem
$P = (x_P : y_P : z_P : t_P)$: Dim2Point
Output: Q: Dim2Point such that $Q = \mathbf{M}(x_P, y_P, z_P, t_P)^T$

$x_P, y_P, z_P, t_P \leftarrow \text{Dim2Coordinates}(P)$
$x_Q \leftarrow M_{0,0} \cdot x_P + M_{0,1} \cdot y_P + M_{0,2} \cdot z_P$
$y_Q \leftarrow M_{1,0} \cdot x_P + M_{1,1} \cdot y_P + M_{1,2} \cdot z_P$
$z_Q \leftarrow M_{2,0} \cdot x_P + M_{2,1} \cdot y_P + M_{2,2} \cdot z_P$
$t_Q \leftarrow M_{3,0} \cdot x_P + M_{3,1} \cdot y_P + M_{3,2} \cdot z_P$
if $t_P \neq 0$ **then**
    $x_Q \leftarrow x_Q + M_{0,3} \cdot t_P$
    $y_Q \leftarrow y_Q + M_{1,3} \cdot t_P$
    $z_Q \leftarrow z_Q + M_{2,3} \cdot t_P$
    $t_Q \leftarrow t_Q + M_{3,3} \cdot t_P$
return $\text{Dim2Point}(x_Q : y_Q : z_Q : t_Q)$

Algorithm 4.121 ProductToTheta — Cost: $15\mathbf{M} + 8\mathbf{a} + (5\mathbf{M} + 4\mathbf{a})$

Input: $\mathbf{M}$: 4×4 matrix of Fp2Elem
P: ECPointPair
Output: Q: Dim2Point // Conversion of P into theta coordinates via $\mathbf{M}$

$P_1, P_2 \leftarrow P$
$x_1, z_1 \leftarrow P_1$
$x_2, z_2 \leftarrow P_2$
$x_Q \leftarrow x_1 \cdot x_2$
$y_Q \leftarrow x_1 \cdot z_2$
$z_Q \leftarrow z_1 \cdot x_2$
if $z_1 \neq 0$ **and** $z_2 \neq 0$ **then**
    $t_Q \leftarrow z_1 \cdot z_2$
    $Q \leftarrow \text{Dim2Point}(x_Q : y_Q : z_Q : t_Q)$
else
    $Q \leftarrow \text{Dim2Point}(x_Q : y_Q : z_Q : 0)$
$Q \leftarrow \text{ApplyMatrix}(\mathbf{M}, Q)$
return Q

4.5.7. Splitting the codomain

Consider a splitting $(2,2)$-isogeny $\varphi\colon A \to B$ with $B = E_3 \times E_4$. Unlike for gluings, there is no specific difficulty in computing and evaluating φ with GenericCodomain and GenericEval. GenericCodomain should return

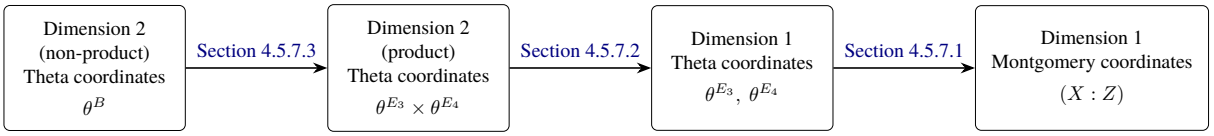
Algorithm 4.122 GluingCodomain — Cost: $118\mathbf{M} + 20\mathbf{S} + 99\mathbf{a} + 16\mathbf{n} + 6\mathbf{i}$

Input: E_1: ECCurve // $\varphi\colon E_1 \times E_2 \to B$
E_2: ECCurve // Coefficient $(A_{24} : C_{24})$ for E_1 and E_2
K_1: ECPointPair
K_2: ECPointPair // $\ker(\varphi) = \langle [4]K_1, [4]K_2 \rangle$
Output: B: Dim2Structure
I_1: Dim2Point // $I_1 = \widetilde{\theta}^B(\varphi(K_1))^{\odot -1}$
I_2: Dim2Point // $I_2 = \widetilde{\theta}^B(\varphi(K_2))^{\odot -1}$
U: Dim2Point // $U = \widetilde{\theta}^B(0_B)^{\odot -1} = (\alpha^{-1} : \beta^{-1} : \gamma^{-1} : 0)$
$\mathbf{M}$: 4×4 matrix of Fp2Elem // Change of coordinates matrix

1: **for** i **from** 1 **up to** 2 **do**
2: $K_{i,1}, K_{i,2} \leftarrow K_i$
3: $K'_{i,1} \leftarrow \mathsf{xDBL}(K_{i,1}, E_1)$
4: $K'_{i,2} \leftarrow \mathsf{xDBL}(K_{i,2}, E_2)$
5: $K'_i \leftarrow \mathsf{ECPointPair}(K'_{i,1}, K'_{i,2})$
6: $\mathbf{M} \leftarrow \mathsf{ThetaChangeOfCoordinates}(K'_1, K'_2)$
7: **for** i **from** 1 **up to** 2 **do**
8: $P_i \leftarrow \mathsf{ProductToTheta}(\mathbf{M}, K_i)$
9: $P_i \leftarrow \mathsf{Squaring}(P_i)$
10: $P_i \leftarrow \mathsf{Hadamard}(P_i)$
11: $x_i, y_i, z_i, t_i \leftarrow \mathsf{Dim2Coordinates}(P_i)$
12: $A \leftarrow x_1 \cdot x_2$
13: $B \leftarrow y_1 \cdot x_2$
14: $C \leftarrow x_1 \cdot z_2$
15: $A^{-1} \leftarrow y_1 \cdot z_2$
16: $B^{-1} \leftarrow C$
17: $C^{-1} \leftarrow B$
18: $x^{-1} \leftarrow z_1 \cdot C^{-1}$
19: $y^{-1} \leftarrow x_1 \cdot A^{-1}$
20: $z^{-1} \leftarrow y_2 \cdot B^{-1}$
21: $t^{-1} \leftarrow x_2 \cdot A^{-1}$
22: $0_B \leftarrow \mathsf{Dim2Point}(A : B : C : 0)$
23: $0_B \leftarrow \mathsf{Hadamard}(0_B)$
24: $B \leftarrow \mathsf{Dim2Structure}(0_B)$
25: $U \leftarrow \mathsf{Dim2Point}(A^{-1} : B^{-1} : C^{-1} : 0)$
26: $I_1 \leftarrow \mathsf{Dim2Point}(x^{-1} : x^{-1} : y^{-1} : y^{-1})$
27: $I_2 \leftarrow \mathsf{Dim2Point}(z^{-1} : t^{-1} : z^{-1} : t^{-1})$
28: **return** $B, I_1, I_2, U, \mathbf{M}$

$\widetilde{\theta}^B(0_B)^{\odot -1}$, from which one can obtain the theta null-point $\theta^B(0_B)$ easily. The question, instead, is: how can we, from this data, derive a system of $(X : Z)$-coordinates and an associated Montgomery equation for E_3 and E_4?

The splitting procedure to do this is summarised as follows, where each step will be detailed in the indicated section.



4.5.7.1. Theta structures on elliptic curves. Let E be a Montgomery curve with coefficient $A_E = -2\frac{a^4+b^4}{a^4-b^4}$. Let θ_0^E be a theta structure on E, defined analogously to the dimension 2 case. In particular, a theta structure on E is a map

$$\theta^E\colon P \in E(\mathbb{F}_{p^2}) \to (\theta_0^E(P) : \theta_1^E(P)) \in \mathbb{P}^1(\mathbb{F}_{p^2}),$$

satisfying certain nice properties.

The conversion formula between the Montgomery model and theta model is given by:

$$(\theta_0^E : \theta_1^E) \mapsto (a(x - z) : b(x + z)) \tag{10}$$

$$(x : z) \leftarrow\!\shortmid (b\theta_0^E + a\theta_1^E : -b\theta_0^E + a\theta_1^E). \tag{11}$$

This conversion formula is canonical, in the sense that it is the one induced by the basis of $E[4]$ formed by $P = (a + b : a - b)$ and $Q = (-1 : 1)$ in Montgomery $(X : Z)$-coordinates.

Given any theta structure on E represented by its theta null-point $(a' : b')$, we fix θ^E to be among the 6 canonical theta structures with theta null-point in

$$\{(a' : b'), (b' : a'), (a' + b' : a' - b'), (a' - b' : a' + b'), (a' + \mathbf{i}b' : a' - \mathbf{i}b'), (a' - \mathbf{i}b' : a' + \mathbf{i}b')\}$$

so that the corresponding Montgomery coefficient A_E is maximal for the order given by Definition 4.2.1. This process is summarised in ThetaToMontgomery. More details are provided in Appendix A.1.3.1.

Algorithm 4.123 GluingEval — Cost: $40\mathbf{M} + 8\mathbf{S} + 36\mathbf{a} + 8\mathbf{a} \cdot \texttt{hadamard}$

Input: $P_1 \pm W_1$: BaryCoords, $(u_1 : v_1 : w_1)$ such that $P_1 - W_1 = (u_1 + v_1 : w_1)$ and $P_1 + W_1 = (u_1 - v_1 : w_1)$
$P_2 \pm W_2$: BaryCoords, $(u_2 : v_2 : w_2)$ such that $P_2 - W_2 = (u_2 + v_2 : w_2)$ and $P_2 + W_2 = (u_2 - v_2 : w_2)$
I: Dim2Point // $I = \widetilde{\theta}^B(\varphi(W))^{\odot -1}$ with $W \leftarrow (W_1, W_2)$
$\mathbf{M}$: 4×4 matrix of Fp2Elem
hadamard : boolean
Output: Q: Dim2Point // $\theta^B(\varphi(P_1, P_2))$ if hadamard $=$ false and $\widetilde{\theta}^B(\varphi(P_1, P_2))$ otherwise

$u_1, v_1, w_1 \leftarrow P_1 \pm W_1$
$u_2, v_2, w_2 \leftarrow P_2 \pm W_2$
$x_U \leftarrow u_1 \cdot u_2$
$t_V \leftarrow v_1 \cdot v_2$
$x_U \leftarrow x_U + t_V$ // $x_U = u_1u_2 + v_1v_2$
$y_U \leftarrow u_1 \cdot w_2$
$z_U \leftarrow w_1 \cdot u_2$
$t_U \leftarrow w_1 \cdot w_2$
$x_V \leftarrow u_1 + v_1$
$y_V \leftarrow u_2 + v_2$
$x_V \leftarrow x_V \cdot y_V$ // $x_V = (u_1 + v_1)(u_2 + v_2)$
$x_V \leftarrow x_V - x_U$ // $x_V = v_1u_2 + u_1v_2$
$y_V \leftarrow v_1 \cdot w_2$
$z_V \leftarrow w_1 \cdot v_2$
$U \leftarrow \text{Dim2Point}(x_U : y_U : z_U : t_U)$
$V \leftarrow \text{Dim2Point}(x_V : y_V : z_V : 0)$
$U \leftarrow \text{ApplyMatrix}(\mathbf{M}, U)$
$V \leftarrow \text{ApplyMatrix}(\mathbf{M}, V)$
$U \leftarrow \text{Squaring}(U)$
$V \leftarrow \text{Squaring}(V)$
$x_U, y_U, z_U, t_U \leftarrow \text{Dim2Coordinates}(U)$
$x_V, y_V, z_V, t_V \leftarrow \text{Dim2Coordinates}(V)$
$x \leftarrow x_U - x_V$
$y \leftarrow y_U - y_V$
$z \leftarrow z_U - z_V$
$t \leftarrow t_U - t_V$
$Q \leftarrow \text{Dim2Point}(x : y : z : t)$
$Q \leftarrow \text{Hadamard}(Q)$
$Q \leftarrow \text{PointwiseMult}(Q, I)$
if hadamard **then**
  return Q
else
  return $\text{Hadamard}(Q)$

Algorithm 4.124 GluingEvalSpecial — Cost: $18\mathbf{M} + 4\mathbf{S} + 28\mathbf{a}$

Input: P: ECPointPair of the form $(P_1, 0)$ or $(0, P_2)$
U: Dim2Point // Codomain inverse dual theta null-point $U = (\alpha^{-1} : \beta^{-1} : \gamma^{-1} : 0)$
$\mathbf{M}$: 4×4 matrix of Fp2Elem // Change of coordinates matrix
Output: $\theta^B(\varphi(P))$: Dim2Point

$Q \leftarrow \text{ProductToTheta}(\mathbf{M}, P)$
$Q \leftarrow \text{Squaring}(Q)$
$Q \leftarrow \text{Hadamard}(Q)$
$x_Q, y_Q, z_Q, t_Q \leftarrow \text{Dim2Coordinates}(Q)$
if $t_Q \neq 0$ **then**
  raise Exception ("GluingEvalSpecial failed: last coordinate should be zero")
$\alpha^{-1}, \beta^{-1}, \gamma^{-1}, 0 \leftarrow \text{Dim2Coordinates}(U)$
$x_Q \leftarrow x_Q \cdot \alpha^{-1}$
$y_Q \leftarrow y_Q \cdot \beta^{-1}$
$z_Q \leftarrow z_Q \cdot \gamma^{-1}$
$Q \leftarrow \text{Dim2Point}(x_Q : y_Q : z_Q : 0)$
return $\text{Hadamard}(Q)$

4.5.7.2. Product theta structures. From two theta structures θ^{E_3} and θ^{E_4} on E_3 and E_4 respectively, we define the *product theta structure* $\theta^{E_3} \times \theta^{E_4}$ on $B = E_3 \times E_4$ as follows:

$$\theta^{E_3} \times \theta^{E_4}(P, Q) \coloneqq (\theta_0^{E_3}(P)\theta_0^{E_4}(Q) : \theta_1^{E_3}(P)\theta_0^{E_4}(Q) : \theta_0^{E_3}(P)\theta_1^{E_4}(Q) : \theta_1^{E_3}(P)\theta_1^{E_4}(Q)),$$

for all $(P, Q) \in E_3 \times E_4$. Conversely, if we denote $\theta^{E_3} \times \theta^{E_4}(P, Q) \coloneqq (x : y : z : t)$, then we have

$$\theta^{E_3}(P) = \begin{cases} (x : y) & \text{if } x \neq 0 \text{ or } y \neq 0 \\ (z : t) & \text{otherwise} \end{cases} \tag{12}$$

$$\theta^{E_4}(Q) = \begin{cases} (x : z) & \text{if } x \neq 0 \text{ or } z \neq 0 \\ (y : t) & \text{otherwise} \end{cases} \tag{13}$$

Algorithm 4.125 ComputeMontCoeff — Cost: $4\mathbf{S} + 3\mathbf{a}$

Input: (a, b): couple of Fp2Elem forming a one-dimensional theta null-point
Output: (A, C): : couple of Fp2Elem which projectively define the Montgomery coefficient

$a_4 \leftarrow a^2$
$a_4 \leftarrow a_4^2$
$b_4 \leftarrow b^2$
$b_4 \leftarrow b_4^2$
$A \leftarrow a_4 + b_4$
$A \leftarrow A + A$ // $A = 2(a^4 + b^4)$
$C \leftarrow b_4 - a_4$ // $C = -(a^4 - b^4)$
if $C == 0$ **then**
  raise Exception ("ComputeMontCoeff failed: Montgomery C coefficient is zero")
return A, C

Algorithm 4.126 ThetaToMontgomery — Cost: $\mathbf{I} + 9\mathbf{M} + 12\mathbf{S} + 16\mathbf{a} + 7\mathbf{n} + 2\mathbf{i}$

Input: (a, b): couple of Fp2Elem forming a one-dimensional theta null-point
Output: E: ECCurve with maximal Montgomery coefficient A_E
$\mathbf{N}$: 2×2 matrix of Fp2Elem to convert theta coordinates into Montgomery $(x : z)$ coordinates

$a' \leftarrow \mathbf{i}a$
$b' \leftarrow \mathbf{i}b$
$a_1 \leftarrow a + b$
$b_1 \leftarrow a - b$
$a_2 \leftarrow a + b'$ // $a_2 = a + \mathbf{i}b$
$b_2 \leftarrow a - b'$ // $b_2 = a - \mathbf{i}b$
$(A_0, C_0) \leftarrow$ ComputeMontCoeff(a, b)
$(A_1, C_1) \leftarrow$ ComputeMontCoeff(a_1, b_1)
$(A_2, C_2) \leftarrow$ ComputeMontCoeff(a_2, b_2)
$C_0, C_1, C_2 \leftarrow$ Fp2BatchInverse$([C_0, C_1, C_2])$
for i **from** 0 **up to** 2 **do**
  $A_i \leftarrow A_i \cdot C_i$
  $A_{i+3} \leftarrow -A_i$
$i_{\max} \leftarrow 0$
$A_{\max} \leftarrow A_0$
for i **from** 1 **up to** 5 **do**
  if Fp2Compare $(A_{\max}, A_i)$ **then**
    $i_{\max} \leftarrow i$
    $A_{\max} \leftarrow A_i$
if $i_{\max} == 0$ **then**
  $\mathbf{N} \leftarrow \begin{pmatrix} b & a \\ -b & a \end{pmatrix}$
else if $i_{\max} == 1$ **then**
  $\mathbf{N} \leftarrow \begin{pmatrix} a & -b \\ b & -a \end{pmatrix}$
else if $i_{\max} == 2$ **then**
  $\mathbf{N} \leftarrow \begin{pmatrix} a & b \\ b' & -a' \end{pmatrix}$
else if $i_{\max} == 3$ **then**
  $\mathbf{N} \leftarrow \begin{pmatrix} b & a \\ b & -a \end{pmatrix}$
else if $i_{\max} == 4$ **then**
  $\mathbf{N} \leftarrow \begin{pmatrix} a & -b \\ -b & a \end{pmatrix}$
else
  $\mathbf{N} \leftarrow \begin{pmatrix} a & b \\ -b' & a' \end{pmatrix}$
return ECCurve$(A_{\max}, 1)$, $\mathbf{N}$

In particular, if $(a : b : c : d)$ is the theta null-point associated to $\theta^{E_3} \times \theta^{E_4}$, then $(a : b)$ and $(a : c)$ are the theta null-points associated to θ^{E_3} and θ^{E_4} respectively. Then it suffices to check that $ad = bc$ to ensure that this theta null-point represents a product theta structure.

4.5.7.3. Splitting change of theta coordinates. Now that we have discussed how to convert a product theta structure into two theta structures on elliptic curves, we need to discuss how to use this to split B into $E_3 \times E_4$. Not all theta structures on B are product theta structures, so we need to first convert it into a product theta structure. In our setting, doing this is particularly easy.

Indeed, assume that $\varphi\colon A \to B = E_3 \times E_4$ is the splitting $(2, 2)$-isogeny of the chain $\Phi\colon E_1 \times E_2 \to E_3 \times E_4$ computed with Isogeny22Chain with kernel inputs given by Equation (28). Let

$$\theta^B(P) \coloneqq (x : y : z : t).$$

Then,

$$\theta^{E_3} \times \theta^{E_4}(P) = (x + y : x - y : z - t : z + t).$$

Equivalently, if in dual coordinates we have $\widetilde{\theta}^B(P) \coloneqq (\widetilde{x} : \widetilde{y} : \widetilde{z} : \widetilde{t})$, then

$$\theta^{E_3} \times \theta^{E_4}(P) = (\widetilde{x} + \widetilde{z} : \widetilde{y} + \widetilde{t} : \widetilde{y} - \widetilde{t} : \widetilde{x} - \widetilde{z}).$$

Using the change of theta coordinates from Lemma A.1.4 and then Equations (12) and (13), we are able to extract from the theta null-point $\theta^B(0_B)$ the theta null-points $\theta^{E_3}(0)$ and $\theta^{E_4}(0)$ with their corresponding Montgomery coefficients A_{E_3} and A_{E_4} and conversion matrices from theta to Montgomery coordinates on E_3 and E_4, respectively. This is implemented in Dim2StructToECCurves. Then, using this data, any Dim2Point $\theta^B(P)$ (*e.g.* obtained by isogeny evaluation) can be converted into its corresponding ECPointPair (P_1, P_2). This can be done with DimTwoPointToECPoints, or SplittingEval to convert an array of points. Note that points given as input to Dim2StructToECCurves, DimTwoPointToECPoints and SplittingEval may be expressed with the dual theta structure $\widetilde{\theta}^B$, which can be indicated by setting the boolean `hadamard` $=$ `true`.

Algorithm 4.127 NonProdToProd — Cost: 4**a**

Input: P: Dim2Point
`hadamard` : boolean
Output: Q: Dim2Point in product theta coordinates

$x, y, z, t \leftarrow$ Dim2Coordinates(P)
if `hadamard` **then**
  $x' \leftarrow x + z$
  $y' \leftarrow y + t$
  $z' \leftarrow y - t$
  $t' \leftarrow x - z$
else
  $x' \leftarrow x + y$
  $y' \leftarrow x - y$
  $z' \leftarrow z - t$
  $t' \leftarrow z + t$
return Dim2Point$(x' : y' : z' : t')$

Algorithm 4.128 Dim2StructToECCurves — Cost: $2\mathbf{I} + 20\mathbf{M} + 24\mathbf{S} + 36\mathbf{a} + 14\mathbf{n} + 4\mathbf{i}$

Input: A: Dim2Structure
`hadamard` : boolean
Output: E_1: ECCurve
E_2: ECCurve // $A \cong E_1 \times E_2$
$\mathbf{N}_1$: 2×2 matrix of Fp2Elem to convert theta coordinates into Montgomery coordinates on E_1
$\mathbf{N}_2$: 2×2 matrix of Fp2Elem to convert theta coordinates into Montgomery coordinates on E_2

$0_A \leftarrow A$
$0'_A \leftarrow$ NonProdToProd$(0_A,$ `hadamard=hadamard`$)$
$a', b', c', d' \leftarrow$ Dim2Coordinates$(0'_A)$
$t_1 \leftarrow a' \cdot d'$
$t_2 \leftarrow b' \cdot c'$
if $a' == 0$ **or** $b' == 0$ **or** $c' == 0$ **or** $t_1 \neq t_2$ **then**
  raise Exception ("Dim2StructToECCurves failed: input not isomorphic to a product of Montgomery curves")
$E_1, \mathbf{N}_1 \leftarrow$ ThetaToMontgomery(a', b')
$E_2, \mathbf{N}_2 \leftarrow$ ThetaToMontgomery(a', c')
return E_1, E_2, $\mathbf{N}_1$, $\mathbf{N}_2$

4.5.8. The $(2,2)$-isogeny chain computation

We have now introduced all the necessary subroutines to construct the Isogeny22Chain algorithm. Similarly to TwoIsogenyChain, Isogeny22Chain uses a divide-and-conquer strategy to reduce the number of point doublings and isogeny evaluations to $O(e\log(e))$ and keeps track of the points and their orders with the arrays `strat_pts` and `orders`. Computing the gluing isogeny $\Phi_1 \colon E_1 \times E_2 \to A_1$ and evaluating the necessary points through it is the most technical part, so we summarize it in a separate auxiliary algorithm called FirstIsogeny.

Remark 4.5.12. Note that instead of $K_1 = [2^{e-1}]T_1$ and $K_2 = [2^{e-1}]T_2$, the images of T_1 and T_2 are used as auxiliary points in GluingEval to evaluate the points $[2^{e-r}]T_1$ and $[2^{e-r}]T_2$ in the list `strat_pts` (for $r \in$ `orders`).

Algorithm 4.129 DimTwoPointToECPoints — Cost: $10\mathbf{M} + 8\mathbf{a}$

Input: P: Dim2Point
$\mathbf{N}_1$: 2×2 matrix of Fp2Elem to convert theta coordinates into Montgomery coordinates on E_1
$\mathbf{N}_2$: 2×2 matrix of Fp2Elem to convert theta coordinates into Montgomery coordinates on E_2
`hadamard` : boolean
Output: Q: ECPointPair

$Q \leftarrow$ NonProdToProd$(P, \texttt{hadamard} = \texttt{hadamard})$
$x, y, z, t \leftarrow$ Dim2Coordinates(Q)
if $x \neq 0$ **then**
  $\theta_{1,0} \leftarrow x$
  $\theta_{1,1} \leftarrow y$
  $\theta_{2,0} \leftarrow x$
  $\theta_{2,1} \leftarrow z$
else if $y \neq 0$ **then**
  $\theta_{1,0} \leftarrow x$
  $\theta_{1,1} \leftarrow y$
  $\theta_{2,0} \leftarrow y$
  $\theta_{2,1} \leftarrow t$
else if $z \neq 0$ **then**
  $\theta_{1,0} \leftarrow z$
  $\theta_{1,1} \leftarrow t$
  $\theta_{2,0} \leftarrow x$
  $\theta_{2,1} \leftarrow z$
else
  raise Exception ("Dim2StructToECCurves failed: zero input")
$t_1 \leftarrow x \cdot t$
$t_2 \leftarrow y \cdot z$
if $t_1 \neq t_2$ **then**
  raise Exception ("Dim2StructToECCurves failed: input is not a product Dim2Structure")
$(N_{1,ij})_{0 \leq i,j \leq 1} \leftarrow \mathbf{N}_1$
$(N_{2,ij})_{0 \leq i,j \leq 1} \leftarrow \mathbf{N}_2$
$x_1 \leftarrow N_{1,00} \cdot \theta_{1,0} + N_{1,01} \cdot \theta_{1,1}$
$z_1 \leftarrow N_{1,10} \cdot \theta_{1,0} + N_{1,11} \cdot \theta_{1,1}$
$x_2 \leftarrow N_{2,00} \cdot \theta_{2,0} + N_{2,01} \cdot \theta_{2,1}$
$z_2 \leftarrow N_{2,10} \cdot \theta_{2,0} + N_{2,11} \cdot \theta_{2,1}$
$Q_1 \leftarrow$ ECPoint(x_1, z_1)
$Q_2 \leftarrow$ ECPoint(x_2, z_2)
return ECPointPair(Q_1, Q_2)

Algorithm 4.130 SplittingEval — Cost: $(10\mathbf{M} + 8\mathbf{a}) \cdot \#\texttt{pts}$

Input: `pts` : array of Dim2Point
$\mathbf{N}_1$: 2×2 matrix of Fp2Elem to convert theta coordinates into Montgomery coordinates on E_1
$\mathbf{N}_2$: 2×2 matrix of Fp2Elem to convert theta coordinates into Montgomery coordinates on E_2
`hadamard` : boolean
Output: `image_pts` : array of ECPointPair

Initialise empty array `image_pts`
for each R in `pts` **do**
  $P \leftarrow$ DimTwoPointToECPoints$(R, \mathbf{N}_1, \mathbf{N}_2, \texttt{hadamard=hadamard})$
  Append P to `image_pts`
return `image_pts`

Indeed, using GluingLadder and BARYComponents, we can obtain the barycentric coordinates $[2^{e-r} \pm 1]T_1$ and $[2^{e-r} \pm 1]T_2$ that we can use as inputs of GluingEval together with T_1 and T_2 in order to evaluate $[2^{e-r}]T_1$ and $[2^{e-r}]T_2$.

Algorithm 4.131 FirstIsogeny

Input: E_1: ECCurve
E_2: ECCurve
T_1: ECPointPair
T_2: ECPointPair // (T_1, T_2) are given by Equation (28)
e: Int // Length of the chain
`pts` : array of ECPointPair of the form $(P_1, 0)$ or $(0, P_2)$
Output: B: Dim2Structure // Codomain of $\Phi_1 : E_1 \times E_2 \to B$
`orders` : array of Int
`strat_pts` : array of couple of Dim2Point // $\Phi_1([2^{e-r}]T_1), \Phi_1([2^{e-r}]T_2)$ for $r \in$ `orders`
`image_pts` : array of Dim2Point // $\Phi_1(P)$ for $P \in$ `pts`
k : length of `strat_pts`

Initialize a list `orders` $\leftarrow [e]$
$T_{1,1}, T_{1,2} \leftarrow T_1$
$T_{2,1}, T_{2,2} \leftarrow T_2$
for $i, j \in \{1, 2\}$ **do**
  Initialize a list `kerij_even` $\leftarrow [T_{i,j}]$
  Initialize a list `kerij_odd` $\leftarrow [0_{E_j}]$
$k \leftarrow 0$
while `orders`$[k] \neq 1$ **do**
  if `orders`$[k] \geq 16$ **then**
    $n \leftarrow \lfloor$`orders`$[k]/2 \rfloor$
  else
    $n \leftarrow$ `orders`$[k] - 1$
  for $i, j \in \{1, 2\}$ **do**
    `kerij_even`$[k+1]$, `kerij_odd`$[k+1] \leftarrow$ GluingLadder(`kerij_even`$[k]$, `kerij_odd`$[k]$, $T_{i,j}, E_j, n$)
      // `kerij_even`$[k] = [2^{e-}$`orders`$^{[k]}]T_{i,j}$ and `kerij_odd`$[k] = [2^{e-}$`orders`$^{[k]} - 1]T_{i,j}$
  `orders`$[k+1] \leftarrow$ `orders`$[k] - n$
  $k \leftarrow k + 1$
$K_1 \leftarrow$ ECPointPair(`ker11_even`$[k]$, `ker12_even`$[k]$)
$K_2 \leftarrow$ ECPointPair(`ker21_even`$[k]$, `ker22_even`$[k]$)
$B, I_1, I_2, U, \mathbf{M} \leftarrow$ GluingCodomain(E_1, E_2, K_1, K_2)
$K_{1,1}, K_{1,2} \leftarrow K_1$
$K_{2,1}, K_{2,2} \leftarrow K_2$
for $i \in \{1, 2\}$ **do**
  for $j \in \{1, 2\}$ **do**
    $T_{i,j} \pm K_{i,j} \leftarrow$ BARYComponents$(T_{i,j}, K_{i,j},$ `kerij_odd`$[k])$
  $I_i \leftarrow$ GluingEval$(T_{i,1} \pm K_{i,1}, T_{i,2} \pm K_{i,2}, I_i, \mathbf{M},$ `hadamard=true`$)$ // $I_i = \widetilde{\theta}^{A_1}(\Phi_1(T_i))$
Initialize a list `strat_pts` $\leftarrow$ [(Hadamard(I_1), Hadamard(I_2))]
$I_1 \leftarrow$ Inverse(I_1)
$I_2 \leftarrow$ Inverse(I_2)
for m **from** 1 **up to** k **do**
  for $i \in \{1, 2\}$ **do**
    for $j \in \{1, 2\}$ **do**
      $K_{i,j} \pm T_{i,j} \leftarrow$ BARYComponents(`kerij_even`$[m], T_{i,j},$ `kerij_odd`$[m])$
    $K_i \leftarrow$ GluingEval$(K_{i,1} \pm T_{i,1}, K_{i,2} \pm T_{i,2}, I_i, \mathbf{M},$ `hadamard=false`$)$ // $K_i = \theta^{A_1}(\Phi_1([2^{e-}$`orders`$^{[m]}]T_i))$
  `strat_pts`$[m] \leftarrow (K_1, K_2)$
  `orders`$[k] \leftarrow$ `orders`$[k] - 1$
`image_pts` $\leftarrow$ [GluingEvalSpecial$(P, U, \mathbf{M}) \mid P \in$ `pts`$]$
$k \leftarrow k - 1$
return B, `orders`, `strat_pts`, `image_pts`, k

4.5.8.1. Computing a $(2, 2)$-isogeny chain constructed from Kani's Lemma. We recall from Section 4.5.4 that Kani's Lemma allows us, given as input one-dimensional isogenies $\varphi_1 : E_1 \to E_3$ and $\varphi_2 : E_3 \to E_2$ between

Algorithm 4.132 Isogeny22Chain

Input: E_1: ECCurve
E_2: ECCurve
T_1: ECPointPair
T_2: ECPointPair // (T_1, T_2) are given by Equation (28)
e: Int // Length of the chain
`pts` : array of ECPointPair of the form $(P_1, 0)$ or $(0, P_2)$
`verify` : boolean // `true` if we use VerifyKernel at every step
Output: E_3: ECCurve
E_4: ECCurve // Codomain of $\Phi\colon E_1 \times E_2 \to E_3 \times E_4$ of kernel $\langle [4]T_1, [4]T_2 \rangle$
`image_pts` : array of ECPointPair // $\Phi(P)$ for $P \in$ `pts`

$B, \texttt{orders}, \texttt{strat\_pts}, \texttt{image\_pts}, k \leftarrow \text{FirstIsogeny}(E_1, E_2, T_1, T_2, e, \texttt{pts})$
for m **from** 1 **up to** $e - 1$ **do**
  if $\texttt{orders}[k] \neq 1$ **then**
    if $m == 1$ **then**
      $U, V \leftarrow \text{ComputeConstants}(B)$
    else
      $B, V \leftarrow \text{ComputeConstantsFromIsoData}(U, x_1, x_2, y_1, y_2)$
  $K_1, K_2 \leftarrow \texttt{strat\_pts}[k]$
  while $\texttt{orders}[k] \neq 1$ **do**
    $n \leftarrow \lfloor \texttt{orders}[k]/2 \rfloor$
    for i **from** 1 **up to** n **do**
      $K_1 \leftarrow \text{Dim2DBL}(K_1, U, V)$
      $K_2 \leftarrow \text{Dim2DBL}(K_2, U, V)$
    $\texttt{strat\_pts}[k+1] \leftarrow (K_1, K_2)$
    $\texttt{orders}[k+1] \leftarrow \texttt{orders}[k] - n$
    $k \leftarrow k + 1$
  if $m < e - 1$ **then**
    $U, x_1, x_2, y_1, y_2 \leftarrow \text{GenericCodomain}(B, K_1, K_2, \texttt{verify\_kernel=verify}, \texttt{hadamard=}(m \neq 1))$
  else
    $U, x_1, x_2, y_1, y_2 \leftarrow \text{GenericCodomain}(B, K_1, K_2, \texttt{verify\_kernel=false}, \texttt{hadamard=}(m \neq 1))$
  $\texttt{strat\_pts} \leftarrow \text{GenericEval}(\texttt{strat\_pts}, U, \texttt{hadamard=}(m \neq 1))$
  $\texttt{image\_pts} \leftarrow \text{GenericEval}(\texttt{image\_pts}, U, \texttt{hadamard=}(m \neq 1))$
  $\texttt{orders} \leftarrow [r - 1 \mid r \in \texttt{orders}]$
  $k \leftarrow k - 1$
$B, \_ \leftarrow \text{ComputeConstantsFromIsoData}(U, x_1, x_2, y_1, y_2)$
$E_3, E_4, \mathbf{N}_3, \mathbf{N}_4 \leftarrow \text{Dim2StructToECCurves}(B, \texttt{hadamard=true})$
$\texttt{image\_pts} \leftarrow \text{SplittingEval}(\texttt{image\_pts}, \mathbf{N}_3, \mathbf{N}_4, \texttt{hadamard=true})$
return E_3, E_4, `image_pts`

elliptic curves E_1, E_2, E_3 such that $\deg(\varphi_1) + \deg(\varphi_2) = 2^e$, to construct a $(2^e, 2^e)$-isogeny $\Phi : E_1 \times E_2 \to E_3 \times E_4$. DimTwoKaniIsogeny implements this construction, using Isogeny22Chain as the main subroutine to compute an HD isogeny as a chain of $(2, 2)$-isogenies.

Algorithm 4.133 DimTwoKaniIsogeny

Input: E_1: ECCurve,
P_1, Q_1: $(\text{ECPoint})^2$ in $E_1[2^{e+2}]$,
E_2: ECCurve,
P_2, Q_2: $(\text{ECPoint})^2$ in $E_2[2^{e+2}]$,
e: Int,
`pts`: array of $(\text{ECPoint})^2$ of the form $(P_1, 0)$ or $(0, P_2)$, // empty if verification
`verify`: Boolean // `true` if run during verification
Output: E_3, E_4: ECCurve
`image_pts` : array of $(\text{ECPoint})^2$,
`valid` : Boolean, `false` if evaluation failed,

1: $T_1 \leftarrow \text{ECPointPair}(P_1, P_2)$
2: $T_2 \leftarrow \text{ECPointPair}(Q_1, Q_2)$
3: **try**
4: $\quad E_3, E_4, \texttt{image\_pts} \leftarrow \text{Isogeny22Chain}(E_1, E_2, T_1, T_2, e, \texttt{pts}, \texttt{verify})$
5: **except**
6: $\quad$ **return** $\bot, \bot, \bot, \texttt{false}$
7: **return** $E_3, E_4, \texttt{image\_pts}, \texttt{true}$

4.6. Ideal-to-isogeny translation

The goal of this module is to translate ideals into isogenies and conversely via the Deuring correspondence. We recall the notations from Section 2.2.1. This section treats the isogenies starting from the curve

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

whose endomorphism ring is isomorphic to

$$\mathcal{O}_0 = \langle 1, \mathbf{i}, \frac{\mathbf{i}+\mathbf{j}}{2}, \frac{1+\mathbf{k}}{2} \rangle.$$

This module introduces two main algorithms:

- IdealToIsogeny translates any left $\mathcal{O}_0$-ideal into the isogeny $\varphi_I : E_0 \to E_I$ associated to I;
- KernelDecomposedToIdeal translates a 2^f-isogeny[6] $E_0 \to E$ into its corresponding left $\mathcal{O}_0$-ideal.

Note that these algorithms deeply rely on the fact that we use E_0 as a starting curve, and will use some precomputed data related to this curve.

4.6.0.1. Precomputation on E_0. Henceforth, we assume that the following elements have been precomputed and are given as implicit parameters to the algorithms introduced in the rest of this section.

(1) $\mathcal{B}_0 = (P_0, Q_0)$: ECBasis_f, a basis of $E_0[2^f]$.
(2) $\mathbf{M}_\mathbf{i}, \mathbf{M}_{\frac{\mathbf{i}+\mathbf{j}}{2}}, \mathbf{M}_{\frac{1+\mathbf{k}}{2}} : \mathbb{M}_2(\mathbb{Z}/2^f\mathbb{Z})$, three matrices describing the action of $\mathbf{i}$, $\frac{\mathbf{i}+\mathbf{j}}{2}$, and $\frac{1+\mathbf{k}}{2}$ on $\mathcal{B}_0$ as in Eq. (2), so that, for example, $\big(\mathbf{i}(P_0), \mathbf{i}(Q_0)\big) = \mathcal{B}_0 \cdot \mathbf{M}_i$.

EndomorphismToMatrix computes the matrix $\mathbf{M}_\beta$ for a given endomorphism $\beta \in \mathcal{O}_0$.

Algorithm 4.134 EndomorphismToMatrix

Input: β: QuatElem, $\beta \in \mathcal{O}_0$
Output: $\mathbf{M}_\beta : \mathbb{M}(\mathbb{Z}/2^f\mathbb{Z})$, the matrix defining the action of the endomorphism β on (P_0, Q_0)

1: $x, y, z, t \leftarrow \text{Coordinates}\mathsf{O}_0(\beta)$
2: $\mathbf{M}_\beta \leftarrow \left(\begin{smallmatrix} x & 0 \\ 0 & x \end{smallmatrix}\right) + y\mathbf{M}_\mathbf{i} + z\mathbf{M}_{(\mathbf{i}+\mathbf{j})/2} + t\mathbf{M}_{(1+\mathbf{k})/2}$
3: **return** $\mathbf{M}_\beta$

[6] given by its kernel

4.6.1. Translating any left $\mathcal{O}_0$-ideal via IdealToIsogeny

IdealToIsogeny takes as input a left $\mathcal{O}_0$-ideal I and returns the isogeny $\varphi_I \colon E_0 \to E_I$ associated to I. The isogeny φ_I is returned as an *efficient representation* in the sense of [Wes24, Definition 1.3], which is convenient for our algorithmic purposes. In practice, IdealToIsogeny returns its codomain E_I and the image $(\varphi_I(P_0), \varphi_I(Q_0))$ of a basis (P_0, Q_0) of $E_0[2^f]$.

IdealToIsogeny proceeds with the Qlapoti method introduced in [BCE+25; PR23] and recently improved in [DLS26]. It first calls a sub-algorithm that we call IdealToIsogenyNormEquation to find two left $\mathcal{O}_0$-ideals $I_1, I_2 \sim I$ of odd norm and such that

$$\mathsf{IdealNorm}(I_1) + \mathsf{IdealNorm}(I_2) = 2^e, \tag{14}$$

with $e \leq f$. For technical reasons related to 2-dimensional isogeny computations, we set $e \coloneqq f - 2$.

The second step of IdealToIsogeny is to compute a $(2^e, 2^e)$-isogeny $E_0^2 \to E_I \times E'$ obtained by applying Kani's lemma [Kan97] (see also [MMP+23, Theorem 1]) as follows. Let $\varphi_1 \colon E_0 \to E_I$ and $\varphi_2 \colon E_0 \to E_I$ be the isogenies associated to I_1 and I_2 respectively and consider a commutative diagram:

$$\begin{array}{ccc} E' & \xrightarrow{\psi_1} & E_0 \\ {\scriptstyle \psi_2}\uparrow & \circlearrowleft & \uparrow{\scriptstyle \widehat{\varphi}_2} \\ E_0 & \xrightarrow{\varphi_1} & E_I \end{array}$$

where ψ_1 and ψ_2 have degrees $N_1 \coloneqq \mathsf{IdealNorm}(I_1)$ and $N_2 \coloneqq \mathsf{IdealNorm}(I_2)$ respectively (these isogenies are meant to make the diagram commute but are never computed). Then Eq. (14) ($N_1 + N_2 = 2^e$) and Kani's lemma ensure the existence of a $(2^e, 2^e)$-isogeny:

$$\Phi \coloneqq \begin{pmatrix} \varphi_1 & \widehat{\varphi}_2 \\ -\psi_2 & \widehat{\psi}_1 \end{pmatrix} : E_0^2 \to E_I \times E',$$

written as a matrix because it acts on points by matrix-vector multiplication on the left as follows:

$$\Phi(P, Q) = (\varphi_1(P) + \varphi_2(Q), -\psi_2(P) + \widehat{\psi}_1(Q)). \tag{15}$$

The 2-dimensional isogeny Φ can be computed from its kernel, given by:

$$\ker(\Phi) = \{([N_1]P, \widehat{\varphi}_2 \circ \varphi_1(P)) \mid P \in E_0[2^e]\}.$$

Knowing I_1 and I_2, it is easy to evaluate $\widehat{\varphi}_2 \circ \varphi_1$ as follows. Indeed, since $I_1, I_2 \sim I$, one can find $\beta_1, \beta_2 \in I$ such that $I_1 = I\overline{\beta_1}/N$ and $I_2 = I\overline{\beta_2}/N$ with $N = \mathsf{IdealNorm}(I)$ (in practice β_1, β_2 are outputs of IdealToIsogenyNormEquation). By the Deuring correspondence (see [Dar25, Lemma 1.2.24]), we then have $\widehat{\varphi}_2 \circ \varphi_1 = \iota_0(\theta)$, where $\theta \coloneqq \beta_2\overline{\beta_1}/N \in \mathcal{O}_0$ and ι_0 is the standard isomorphism $\mathcal{O}_0 \xrightarrow{\sim} \mathrm{End}(E_0)$. We then have $\ker(\Phi) = \langle [4]K_1, [4]K_2 \rangle$, where $K_1 \coloneqq ([N_1]P_0, \iota_0(\theta)(P_0))$ and $K_2 \coloneqq ([-N_2]Q_0, \iota_0(\theta)(Q_0))$ form an isotropic subgroup of $E_0^2[2^f]$. The points K_1 and K_2 are easy to evaluate with EndomorphismToMatrix. Then Φ can be computed from the inputs E_0, K_1 and K_2 with Isogeny22Chain.

We now explain how to obtain the desired representation of φ_I from Φ. One can easily extract E_I from the codomain of Φ output by Isogeny22Chain. It remains to explain how to obtain $(\varphi_I(P_0), \varphi_I(Q_0))$. Using the fact that $I_1 = I\overline{\beta_1}/N$ and [Dar25, Lemma 1.2.24], we get that $[N_1]\varphi_I = \varphi_1 \circ \iota_0(\beta_1)$, so we obtain

$$\varphi_I(P_0) = [\nu_1]\varphi_1 \circ \iota_0(\beta_1)(P_0) \quad \text{and} \quad \varphi_I(Q_0) = [\nu_1]\varphi_1 \circ \iota_0(\beta_1)(Q_0),$$

where $\nu_1 \equiv N_1^{-1} \mod 2^f$ and $\varphi_1(P_0), \varphi_1(Q_0)$ can be obtained from Equation (15) with $P = P_0$ or $P = Q_0$, and $Q = 0$.

4.6.2. The norm equations and prerequisites

This section deals with the norm equation part of our IdealToIsogeny algorithm. The final goal of this section is to describe the IdealToIsogenyNormEquation algorithm which is exactly the Qlapoty improvement to the original Qlapoti algorithm introduced in [DLS26].

4.6.2.1. Two-dimensional lattices. This section deals with two-dimensional lattices, which are identified with 2×2 integer matrices. First, we define a type for $\mathbb{Z}^2$.

Definition 4.6.1. The type VecTwo consists of a pair (a, b) of Int.

The natural operations on Int naturally extend coordinate-wise to the elements of VecTwo. Moreover, we define the additional operations:

Algorithm 4.135 Norm(VecTwo) : Int

$$(a, b) \mapsto \quad a^2 + b^2$$

Algorithm 4.136 InnerProduct(VecTwo, VecTwo) : Int

$$(a, b), (c, d) \mapsto \quad ac + bd$$

Definition 4.6.2. The type LatTwo consists of the tuple a, b, c, d of Int representing the matrix $\begin{pmatrix} a & b \\ c & d \end{pmatrix}$

The usual vector-matrix multiplication of v : VecTwo and M : LatTwo is written $v \cdot M$. We also define the following operations:

Algorithm 4.137 Determinant(LatTwo) : Int

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} \mapsto \quad ad - bc$$

Algorithm 4.138 Comatrix(LatTwo) : LatTwo

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} \mapsto \quad \begin{pmatrix} d & -b \\ -c & a \end{pmatrix}$$

Algorithm 4.139 LatToBasis(LatTwo) : VecTwo, VecTwo

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} \mapsto \quad (a, b), (c, d)$$

Algorithm 4.140 BasisToLat(VecTwo, VecTwo) : LatTwo

$$(a, b), (c, d) \mapsto \quad \begin{pmatrix} a & b \\ c & d \end{pmatrix}$$

4.6.2.2. Main loops. Following the presentation made in [DLS26], the IdealToIsogenyNormEquation algorithm can be divided in two main loops that we present below as the LoopOne and LoopTwo algorithms. There is also a DivideMinima algorithm that will be required in some special case to ensure that the LoopOne algorithm can be run smoothly.

4.6.2.3. The norm equation algorithm. We now have all the necessary building blocks to introduce the IdealToIsogenyNormEquation algorithm. It essentially consists of running the LoopTwo sub-algorithm on the output of the LoopOne step. If no solution is found, we simply run LoopOne again to rerandomize the input to LoopTwo.

Algorithm 4.141 Compute2DLattice

Input: N: Int positive
α: QuatElem with $\mathsf{GCD}(\mathsf{Trace}(\alpha), \mathsf{Trace}(\mathbf{i}\alpha), 2N) = 1$
Output: Λ: LatTwo, kernel lattice of $(s,t) \mapsto \mathsf{Trace}(\overline{(s+\mathbf{i}t)} \cdot \alpha) \mod 4N$
(s_1, t_1): VecTwo with $\mathsf{Trace}(\overline{(s_1+\mathbf{i}t_1)} \cdot \alpha) = 1 \mod 4N$

1: $a \leftarrow \mathsf{Trace}(\alpha)$
2: $b \leftarrow -\mathsf{Trace}(\mathbf{i}\alpha)$
3: $g_a \leftarrow \mathsf{GCD}(a, 4N)$
4: $a' \leftarrow a/g_a$
5: $c \leftarrow (a'^{-1} \mod (4N/g_a))$
6: $c \leftarrow -b \cdot c \mod 4N/g_a$
7: $\Lambda \leftarrow \begin{pmatrix} c & g_a \\ 4N/g_a & 0 \end{pmatrix}$
8: $k, u, v \leftarrow \mathsf{XGCD}(a, b)$
9: $q, w, \square \leftarrow \mathsf{XGCD}(k, 4N)$
10: **return** $\Lambda, (wu \mod 4N, wv \mod 4N)$

Algorithm 4.142 ComputeLatticeAndReduce

Input: N: Int positive
α: QuatElem with $\mathsf{GCD}(\mathsf{Trace}(\alpha), \mathsf{Trace}(\mathbf{i}\alpha), 2N) = 1$
Output: Λ: LatTwo, reduced basis of the kernel lattice of $(s,t) \mapsto \mathsf{Trace}(\overline{(s+\mathbf{i}t)} \cdot \alpha) \mod 4N$
(s_1, t_1): VecTwo with $\mathsf{Trace}(\overline{(s_1+\mathbf{i}t_1)} \cdot \alpha) = 1 \mod 4N$

1: $a \leftarrow \mathsf{Trace}(\alpha)$
2: $b \leftarrow -\mathsf{Trace}(\mathbf{i}\alpha)$
3: $g_a \leftarrow \mathsf{GCD}(a, 4N)$
4: $a' \leftarrow a/g_a$
5: $c \leftarrow (a'^{-1} \mod (4N/g_a))$
6: $c \leftarrow -b \cdot c \mod 4N/g_a$
7: $A, B, C, D \leftarrow \mathsf{LatReduction2DZ}(4N/g_a, c, g_a, \mathsf{BitLenNormEqInput2DLat})$
8: $\Lambda \leftarrow \begin{pmatrix} A & C \\ B & D \end{pmatrix}$
9: $k, u, v \leftarrow \mathsf{XGCD}(a, b)$
10: $q, w, \square \leftarrow \mathsf{XGCD}(k, 4N)$
11: **return** $\Lambda, (wu \mod 4N, wv \mod 4N)$

Algorithm 4.143 ComputeSmallSolutions

Input: N: Int, positive
Λ: LatTwo, kernel lattice of $(s,t) \mapsto as + bt \mod 4N$
λ: Int with $\mathsf{GCD}(\lambda, 2N) = 1$
(s_1, t_1): VecTwo with $as_1 + bt_1 = 1 \mod 4N$
d: Int
Output: (s,t): VecTwo smallest (s,t) such that $\lambda(as+bt) = d \mod 4N$

1: $\mu \leftarrow \mathsf{ModularInverse}(\lambda, 4N)$
2: $v_t \leftarrow (-\mu \cdot d \cdot s_1 \mod 4N, -\mu \cdot d \cdot t_1 \mod 4N)$
3: $v_c \leftarrow \left\lfloor \frac{v_t \cdot \mathsf{Comatrix}(\Lambda)}{\mathsf{Determinant}(\Lambda)} \right\rceil \cdot \Lambda$ // Babai's rounding algorithm
4: $(s,t) \leftarrow v_c - v_t$
5: **return** s, t

Algorithm 4.144 OneStepReduction

Input: v_0, v_1 : VecTwo with $\mathsf{Norm}(v_0) \leq \mathsf{Norm}(v_1)$
Output: v_1' : VecTwo with $\mathsf{Norm}(v_1') \leq \mathsf{Norm}(v_1)$ and $\langle v_0, v_1 \rangle = \langle v_0, v_1' \rangle$

1: $n_0 \leftarrow \mathsf{Norm}(v_0)$
2: $m \leftarrow \mathsf{InnerProduct}(v_0, v_1)$
3: $q \leftarrow \left\lfloor \frac{m}{n_0} \right\rceil$
4: **return** $v_1 - qv_0$

Algorithm 4.145 ReduceGivenMinima

Input: Λ : LatTwo
ω : VecTwo, with $\omega \in \Lambda$ of smallest norm
Output: Λ' : LatTwo representing the reduced basis of the lattice Λ

1: $(a_1, a_2) \leftarrow \omega \cdot \mathsf{Comatrix}(\Lambda)/\mathsf{Determinant}(\Lambda)$ // a_1, a_2 should be coprime integers
2: $1, a_4, -a_3 \leftarrow \mathsf{XGCD}(a_1, a_2)$ // $a_1 a_4 - a_2 a_3 = 1$
3: $v_2 \leftarrow (a_3, a_4) \cdot \Lambda$ // ω, v_2 is another basis of Λ
4: $v_2 \leftarrow \mathsf{OneStepReduction}(\omega, v_2)$
5: **return** $\mathsf{BasisToLat}(\omega, v_2)$

Algorithm 4.146 LoopOne

Input: N: Int, positive
β^I: QuatElem with $\beta^I \in \mathcal{O}_0$
a, b: Int with $0 \leq a < b$
(s_0, t_0): VecTwo smallest nonzero element in the kernel lattice of Trace$(\overline{(s+\mathbf{i}t)}\beta^I) \mod 4N$
$\mathsf{prng}_{\mathsf{def}}$: PRNG
Output: α_0: QuatElem with $\alpha_0 \in \mathcal{O}_0\langle\beta^I, N\rangle$
Λ: LatTwo reduced kernel lattice of Trace$(\overline{(s+\mathbf{i}t)}\alpha_0) \mod 4N$, with its first minimum at least $\sqrt{N/2}$
(s_1, t_1): VecTwo with Trace$(\overline{(s_1+\mathbf{i}t_1)}\alpha_0) = 1 \mod 4N$

$\ell \leftarrow \mathsf{GCD}(s_0, t_0)$
$\omega \leftarrow s_0 + \mathbf{i}t_0$
while `true` **do** // Loop 1
  if $b < 5$ **then**
    $c_3, c_4 \leftarrow 1, 0$
  else
    $c_3 \leftarrow \mathsf{SampleFromInterval}\left(\lceil\sqrt{a/2}\rceil, \lfloor\sqrt{b}\rfloor, \mathsf{prng}_{\mathsf{def}}\right)$
    if $\lceil\sqrt{\max(0, a-c_3^2)}\rceil > \lfloor\sqrt{b-c_3^2}\rfloor$ **then**
      continue
    $c_4 \leftarrow \mathsf{SampleFromInterval}\left(\lceil\sqrt{\max(0, a-c_3^2)}\rceil, \lfloor\sqrt{b-c_3^2}\rfloor, \mathsf{prng}_{\mathsf{def}}\right)$
  if $c_3 == c_4 \mod 2$ **or** $\mathsf{GCD}(c_3, c_4) \neq 1$ **then**
    continue
  if $\sqrt{b} - \sqrt{a} \geq \log(N)^2$ **then**
    if $\gcd(c_3^2 + c_4^2, N) \neq 1$ **then**
      continue
  else
    if $\gcd(c_3^2 + c_4^2, \ell) \neq 1$ **then**
      continue
  $z_c \leftarrow c_3 + \mathbf{i}c_4$
  while $\mathsf{GaussGCD}(\overline{z_c}, \omega) \notin \{\pm 1, \pm\mathbf{i}\}$ **do**
    $\gamma \leftarrow \mathsf{GaussGCD}(\overline{z_c}, \omega)$
    $z_c \leftarrow z_c \cdot \gamma^2 / \mathsf{Norm}(\gamma)$ // $n(\gamma) = \gamma\overline{\gamma}$
  $\alpha_0 \leftarrow z_c \cdot \beta^I$
  $a, b, c, d, e \leftarrow \mathsf{CoordinatesB}(\alpha_0)$ // e must be equal to 2
  $a \leftarrow a \mod 4N$
  $b \leftarrow b \mod 4N$
  $\alpha_0 \leftarrow (a + \mathbf{i}b + \mathbf{j}c + \mathbf{k}d)/2$
  $\Lambda, v_1 \leftarrow \mathsf{Compute2DLattice}(N, \alpha_0)$ // Λ unreduced at this point
  $(s_0 + \mathbf{i}t_0) \leftarrow z_c \cdot (s_0 + \mathbf{i}t_0)$ // (s_0, t_0) should be the smallest nonzero element of Λ
  $\Lambda \leftarrow \mathsf{ReduceGivenMinima}(\Lambda, (s_0, t_0))$
  return α_0, Λ, v_1

Algorithm 4.147 DivideMinima

Input: N: Int, positive
β^I: QuatElem with $\beta^I \in \mathcal{O}_0$
a, b: Int with $0 < a < b$
(s_0, t_0): VecTwo smallest element in the kernel lattice of Trace$(\overline{(s + \mathbf{i}t)}\beta^I) \mod 4N$
Output: γ^I: QuatElem with $\gamma^I \in \mathcal{O}_0\langle\beta^I, N\rangle$
a', b': Int with $0 < a' < b'$
(s'_0, t'_0): VecTwo smallest element in the kernel lattice of Trace$(\overline{(s + \mathbf{i}t)}\gamma^I) \mod 4N$

$\ell^I \leftarrow \mathsf{GCD}(s_0, t_0)$
Let g be the smallest integer such that $5^g > \log^4 N$
$5^h \leftarrow \mathsf{GCD}(5^g, \ell^I)$
$s'_0, t'_0 \leftarrow s_0/5^h, t_0/5^h$
if $2s'_0 - t'_0 \neq 0 \mod 5$ **then** // $2 - \mathbf{i}$ does not divide $s'_0 + \mathbf{i}t'_0$
    $(s'_0 + \mathbf{i}t'_0) \leftarrow (2 + \mathbf{i})^h \cdot (s'_0 + \mathbf{i}t'_0), \gamma^I \leftarrow (2 + \mathbf{i})^h \cdot \beta^I$
else
    $s'_0 + \mathbf{i}t'_0 \leftarrow (2 - \mathbf{i})^h \cdot (s'_0 + \mathbf{i}t'_0), \gamma^I \leftarrow (2 - \mathbf{i})^h \cdot \beta^I$
$a' \leftarrow 5^h \cdot a, b' \leftarrow 5^h \cdot b$
return $\gamma^I, a', b', (s'_0, t'_0)$

Algorithm 4.148 LoopTwo

Input: e: Int, positive
N: Int, positive
α_0: QuatElem, with $\alpha_0 \in \mathcal{O}_0 \setminus \mathbb{Z}[\mathbf{i}, \mathbf{j}]$ and $\mathsf{Norm}(\alpha_0) = 0 \mod N$
Λ: LatTwo, kernel lattice of $(s,t) \mapsto \mathsf{Trace}(\overline{(s+\mathbf{i}t)}\alpha_0) \mod 4N$
(s_1, t_1): VecTwo with $\mathsf{Trace}(\overline{(s_1+\mathbf{i}t_1)}\alpha_0) = 1 \mod 4N$
Output: β_1, β_2: QuatElem, with $\beta_1, \beta_2 \in \mathcal{O}_0\langle\alpha_0, N\rangle$ and $\mathsf{Norm}(\beta_1) + \mathsf{Norm}(\beta_2) = N2^e$

$r_0 \leftarrow \mathsf{Norm}(\alpha_0)/N$
$\delta \leftarrow 1$
if $r_0 == 0 \mod 2$ **then**
  $\delta \leftarrow 3$
$\mu \leftarrow -1$ // test odd solution only
while $\mu < \sqrt{2^{e-2}/r_0}$ **do** // Loop 2
  $\mu \leftarrow \mu + 2$
  if $\gcd(2N, \mu) \neq 1$ **then**
    continue
  $d \leftarrow 2^e - 2\mu^2 r_0 - \mu N$
  $(s,t) \leftarrow \mathsf{ComputeSmallSolutions}(N, \Lambda, \mu, (s_1,t_1), d)$ // (s,t) such that $\mathsf{Trace}(\overline{(s+\mathbf{i}t)}\alpha_0) \equiv 2^e - 2\mu^2 r_0 - \delta N \pmod{4N}$
  if $s \equiv t \pmod 2$ **then**
    continue
  $a \leftarrow \mathsf{Trace}(\alpha_0), b \leftarrow -\mathsf{Trace}(\mathbf{i}\alpha_0)$
  $k \leftarrow (2^e - 2\mu^2 r_0 - \mu a s - \mu b t)/N$
  $z \leftarrow 2k - s^2 - t^2$
  if $z < 0$ **then**
    continue
  sol $\leftarrow \mathsf{Cornacchia}(z)$
  if sol $== \perp$ **then**
    continue
  $(s_c, t_c) \leftarrow$ sol
  if $s_c \not\equiv s \pmod 2$ **then** swap s_c, t_c
  $a_1 \leftarrow (s_c + s)/2, \quad b_1 \leftarrow (t_c + t)/2$
  $a_2 \leftarrow s - a_1, \quad b_2 \leftarrow t - b_1$
  $\beta_1 \leftarrow N(a_1 + \mathbf{i}b_1) + \mu\alpha_0, \quad \beta_2 \leftarrow N(a_2 + \mathbf{i}b_2) + \mu\alpha_0$
  return β_1, β_2
return $\perp$ // Ensure $z = 1 \mod 4$

Algorithm 4.149 IdealToIsogenyNormEquation

Input: J : InertIdeal // left $\mathcal{O}_0$-ideal
e : Int
$\mathsf{prng}_{\mathsf{def}}$: PRNG
Output: β_1 : QuatElem, contained in J // Suitable solution of $\mathrm{nrd}(\beta_1) + \mathrm{nrd}(\beta_2) = 2^e \cdot \mathsf{IdealNorm}(J)$
d_1 : Int // $d_1 = \mathrm{nrd}(\beta_1)/\mathsf{IdealNorm}(J)$
θ : QuatElem // $\theta = \beta_2\overline{\beta_1}/\mathsf{IdealNorm}(J)$

$(N, (x, y)), \gamma \leftarrow \mathsf{SmallestEquivalentIdeal}(J, \mathsf{prng}_{\mathsf{def}})$
$B_1^2(N) \leftarrow N/2, \quad B_M^2(N) \leftarrow 5N/2, \quad B_\alpha^2(N) \leftarrow N(2^{e+3} - 16N)/p$
$\beta^I \leftarrow x + \mathbf{i}y + \frac{\mathbf{i}+\mathbf{j}}{2}$
$\Lambda, \square \leftarrow \mathsf{ComputeLatticeAndReduce}(N, \beta^I)$
$v_0, \square \leftarrow \mathsf{LatToBasis}(\Lambda)$
$\lambda_1^I \leftarrow \mathsf{Norm}(v_0)$
if $\lambda_1^I \leq 2B_1^2(N)/B_\alpha^2(N)$ **then**
    raise Exception ("IdealToIsogenyNormEquation failed: the input ideal is among the bad inputs") // see Section 9.4
$a \leftarrow B_1^2(N)/\lambda_1^I$
$b \leftarrow \min(B_\alpha^2(N)/2, B_M^2(N)/\lambda_1^I)$
if $\sqrt{b} - \sqrt{a} \leq \log^2(N)$ **then**
    $\beta^I, a, b, v_0 \leftarrow \mathsf{DivideMinima}(N, \beta^I, a, b, v_0)$
while `true` **do**
    $\alpha_0, \Lambda, v_1 \leftarrow \mathsf{LoopOne}(N, \beta^I, a, b, v_0, \mathsf{prng}_{\mathsf{def}})$
    $\mathsf{sol} \leftarrow \mathsf{LoopTwo}(e, N, \alpha_0, \Lambda, v_1)$
    if $\mathsf{sol} \neq \perp$ **then**
        $\beta_1, \beta_2 \leftarrow \mathsf{sol}$
        $d_1 \leftarrow \mathsf{Norm}(\beta_1)/N$
        $\theta \leftarrow \mathsf{Mul}(\beta_2, \mathsf{Conjugate}(\beta_1))/N$
        $\beta_1 \leftarrow \mathsf{Mul}(\beta_1, \gamma)/N$
        return β_1, d_1, θ

4.6.2.4. The ideal-to-isogeny algorithm. Now that the IdealToIsogenyNormEquation sub-algorithm has been specified, we are ready to detail the full ideal-to-isogeny translation.

Algorithm 4.150 IdealToIsogeny

Input: I: InertIdeal // left $\mathcal{O}_0$-ideal
$\mathsf{prng}_{\mathsf{def}}$: PRNG
Output: E_I: ECCurve // Codomain of $\varphi_I : E_0 \to E_I$ corresponding to I
$\mathcal{B}_I = (\varphi_I(P_0), \varphi_I(Q_0), \varphi_I(P_0 - Q_0))$: $\mathsf{ECBasis}_f$ // Image of the basis $\mathcal{B}_0$ of $E_0[2^f]$ via $\varphi_I : E_0 \to E_I$

$\beta_1, d_1, \theta \leftarrow \mathsf{IdealToIsogenyNormEquation}(I, f-2, \mathsf{prng}_{\mathsf{def}})$
$d_2 \leftarrow 2^{f-2} - d_1$
$\nu_1 \leftarrow \mathsf{ModularInverse}(d_1, 2^f)$
$\nu_2 \leftarrow \mathsf{ModularInverse}(d_2, 2^f)$
$c_2 \leftarrow \nu_1 \cdot (1 - \nu_2 \cdot 2^{f-2}) \mod 2^f$
$\mathbf{M}_\theta \leftarrow \mathsf{EndomorphismToMatrix}(\theta)$
$(T_{1,2}, T_{2,2}, \square) \leftarrow \mathsf{ActOnBasis}(\mathcal{B}_0, \mathbf{M}_\theta \cdot \left(\begin{smallmatrix} \nu_1 & 0 \\ 0 & c_2 \end{smallmatrix}\right))$
$\mathbf{M}_{\beta_1} \leftarrow \mathsf{EndomorphismToMatrix}(\beta_1)$
$(P_1, Q_1, R_1) \leftarrow \mathsf{ActOnBasis}(\mathcal{B}_0, \nu_1 \cdot \mathbf{M}_{\beta_1})$
$U_1 \leftarrow \mathsf{ECPointPair}(P_1, 0)$
$V_1 \leftarrow \mathsf{ECPointPair}(Q_1, 0)$
$W_1 \leftarrow \mathsf{ECPointPair}(R_1, 0)$
$E_I, \square, (U_I, V_I, W_I), \square \leftarrow \mathsf{DimTwoKaniIsogeny}(E_0, (P_0, Q_0), E_0, (T_{1,2}, T_{2,2}), f-2, (U_1, V_1, W_1), \texttt{false})$
$P_I, \square \leftarrow U_I$
$Q_I, \square \leftarrow V_I$
$R_I, \square \leftarrow W_I$
return $E_I, (P_I, Q_I, R_I)$

4.6.2.5. Kernel-to-ideal translation. For the challenge computation, we are going to go the other way around, and compute an ideal from the kernel of the corresponding isogeny. This gives the KernelDecomposedToIdeal algorithm below.

Algorithm 4.151 $\mathsf{KernelDecomposedToIdeal}(e, c_1, c_2)$

Input: e: Int, positive
c_1, c_2: Int, defining a point $[c_1]P_0 + [c_2]Q_0$ on E_0 of order 2^e generating the kernel of an isogeny φ
Output: I_φ: $\mathsf{GenericO_0Ideal}$, the left $\mathcal{O}_0$-ideal of norm 2^e corresponding to φ

$\mathbf{M}_\alpha \leftarrow \mathbf{M_j} + \mathbf{M}_{\frac{1+\mathbf{k}}{2}}$
$[d_1, d_2]^T \leftarrow \mathbf{M}_\alpha [c_1, c_2]^T$
$\mathbf{M} \leftarrow \begin{pmatrix} c_1 & d_1 \\ c_2 & d_2 \end{pmatrix}$
$[a, b]^T \leftarrow \mathbf{M}^{-1} \mathbf{M_i} [c_1, c_2]^T$
$\alpha \leftarrow a + b\left(\mathbf{j} + \frac{1+\mathbf{k}}{2}\right) - \mathbf{i}$
$\alpha' \leftarrow \mathsf{Mul}(\alpha, 1 + \mathbf{i})$
$t, x, y, z \leftarrow \mathsf{CoordinatesO_0}(\alpha')$
$\gamma \leftarrow 1$
$e' \leftarrow e$
if $t = 0 \mod 2$ **and** $x = 0 \mod 2$ **and** $y = 0 \mod 2$ **and** $z = 0 \mod 2$ **then**
  $\gamma \leftarrow (1 - \mathbf{i})$
  $\alpha \leftarrow \alpha'/2$
  $e' \leftarrow e' - 1$
$I \leftarrow \mathsf{InertGenToIdealPowTwo}(e', \alpha)$
return (γ, I)

4.7. Precomputed values

General Scheme Parameters. The following table gives the constants used as general scheme parameters.

TABLE 8. Global parameters

Constant	NIST Level			Formula
	I	III	V	
λ	128	192	256	
p	See Chapter 5			
f	324	500	664	$\mathsf{DyadicValuation}(p+1)$
D_{mix}	See formula			$\texttt{next\_prime}(p \cdot 2^{2\lambda})$
e_{rsp}	196	302	400	$1 + \lceil \frac{1}{4}(\lambda + 2 \cdot \log_2(p)) \rceil$
D_{rsp}	2^{195}	2^{301}	2^{399}	$2^{e_{\mathsf{rsp}}-1}$

Quaternion Computations. Table 9 gives the constants used in quaternion computations: RandomIdealGivenNorm, EquivalentCoprimeIdeal, UniformO$_0$ClassSampling. Precise requirements on RICofactor and EIBox can be found in Section 9.3 while a precise requirement on D_{mix} can be found in Section 8.1. Table 10 gives the constants used in lattice reduction. Most of them are as defined Table 7. We also provide the three matrices $\mathbf{M}_{\mathbf{i}}, \mathbf{M}_{\frac{\mathbf{i}+\mathbf{j}}{2}}, \mathbf{M}_{\frac{1+\mathbf{k}}{2}} : \mathbb{M}_2(\mathbb{Z}/2^f\mathbb{Z})$, discussed in Section 4.6, describing the action of $\mathbf{i}$, $\frac{\mathbf{i}+\mathbf{j}}{2}$, and $\frac{1+\mathbf{k}}{2}$ on the basis $\mathcal{B}_0 = (P_0, Q_0)$ of $E_0[2^f]$ provided in Section 4.7. Given an endomorphism $\alpha \in \{\mathbf{i}, \frac{\mathbf{i}+\mathbf{j}}{2}, \frac{1+\mathbf{k}}{2}\}$ and a point $R \in \{P_0, Q_0\}$, we provide $a_\alpha(R)$ and $b_\alpha(R)$ such that $\alpha(R) = a_\alpha(R)P_0 + b_\alpha(R)Q_0$.

TABLE 9. Global constants for quaternions

Constant	NIST Level			Formula
	I	III	V	
D_{mix}		See formula		$\texttt{next\_prime}(p \cdot 2^{2\lambda})$
EIBox	15	19	22	See Section 9.3 and Algorithm B.2
EIBoundIteration		See formula		$(\mathsf{EIBox}+1)^4$
BoundIterationRBE	1490	2235	2979	$\lceil 11.636\lambda \rceil$, see Section 9.5
RICofactor	$2^{146}+85$	$2^{220}+217$	$2^{286}+43$	See Section 9.3 and Algorithm B.3

The following table gives the constants used in lattice reduction. Most of them are as defined in Table 7.

NIST-I:

$$a_{\mathbf{i}}(P_0) = \texttt{0xaa13b93638dd194fd68b1539d6abf5d9b354d65dc452cb133526024d24f509054322beb5e60469e6d}$$

$$b_{\mathbf{i}}(P_0) = \texttt{0x6f28bcabec2b4a7959ccb5ba67dfe600d895b73a5d9601a7872106a16d914e980822b261335eac3c7}$$

$$a_{\mathbf{i}}(Q_0) = \texttt{0x72880b5ef20ea83fdea5ca9028ef5f98c46323737ad5faaee70a53f7c91113c33ec69680df262c1ba}$$

$$b_{\mathbf{i}}(Q_0) = \texttt{0x55ec46c9c722e6b02974eac629540a264cab29a23bad34eccad9fdb2db0af6fabcdd414a19fb96193}$$

$$a_{\frac{\mathbf{i}+\mathbf{j}}{2}}(P_0) = \texttt{0x6a13b93638dd194fd68b1539d6abf5d9b354d65dc452cb133526024d24f509054322beb5e60469e6d}$$

$$b_{\frac{\mathbf{i}+\mathbf{j}}{2}}(P_0) = \texttt{0x2f28bcabec2b4a7959ccb5ba67dfe600d895b73a5d9601a7872106a16d914e980822b261335eac3c7}$$

$$a_{\frac{\mathbf{i}+\mathbf{j}}{2}}(Q_0) = \texttt{0x6b2c1612e9a155d2354a0c298fbda84a8ac3bc3e79f353710cb265402e5932993e}$$

ef75c3f1776cfb1

$$b_{\frac{\mathbf{i}+\mathbf{j}}{2}}(Q_0) = \texttt{0x95ec46c9c722e6b02974eac629540a264cab29a23bad34eccad9fdb2db0af6fabc}$$
$$\texttt{dd414a19fb96193}$$

$$a_{\frac{1+\mathbf{k}}{2}}(P_0) = \texttt{0x400000000000000000000000000000000000000000000000000000000000000000000000}$$
$$\texttt{000000000000000}$$

$$b_{\frac{1+\mathbf{k}}{2}}(P_0) = \texttt{0x800000000000000000000000000000000000000000000000000000000000000000000000}$$
$$\texttt{000000000000000}$$

$$a_{\frac{1+\mathbf{k}}{2}}(Q_0) = \texttt{0x956eabfde7e1992ee8479e670e9e64c686286686f98320cf2dd0fa934fa70b3118}$$
$$\texttt{32ad4399bb2642b}$$

$$b_{\frac{1+\mathbf{k}}{2}}(Q_0) = \texttt{0xc000000000000000000000000000000000000000000000000000000000000000000000000}$$
$$\texttt{000000000000001}$$

NIST-III:

$$a_{\mathbf{i}}(P_0) = \texttt{0x19df28f2d7c316d855ea0387774de81d709d04122bb6b5af370731324d939a71e8de}$$
$$\texttt{3d5c3f3dcd5435decd5b34fd7df75c4974c8e9f38892f96a886b93337}$$

$$b_{\mathbf{i}}(P_0) = \texttt{0x7b5cb2bffd3cdad392e812fddb0ad3069068975a3bbad3a9d2d541657a4135bb87a7}$$
$$\texttt{d28fb0a4bfbe40058ee7e771e81ba39cb86cd2f81d85637b2429e45b1}$$

$$a_{\mathbf{i}}(Q_0) = \texttt{0x89c9cf0b50897b57bf5123c0373656bc39bab67872b0b2aa634d9ffe5b079a8fe17d}$$
$$\texttt{e4982bb45ebb9c041d980d025716c40df3b18e4122061e1c20cfd028e}$$

$$b_{\mathbf{i}}(Q_0) = \texttt{0xe620d70d283ce927aa15fc7888b217e28f62fbedd4494a50c8f8cecdb26c658e1721}$$
$$\texttt{c2a3c0c232abca2132a4cb028208a3b68b37160c776d069577946ccc9}$$

$$a_{\frac{\mathbf{i}+\mathbf{j}}{2}}(P_0) = \texttt{0xb95bb56bf47896da506ed42f9ee4b4b788229e897f4106826462b3d1a3bed13df9}$$
$$\texttt{0bb1a7d8b6265177e3a0cba4ff6c32cd8a625fba936ba3fb0277f4caa15}$$

$$b_{\frac{\mathbf{i}+\mathbf{j}}{2}}(P_0) = \texttt{0x1c8f9bc081cb2a250d69b37c95fc3a287009b6cd26ad413a4e4d12f392b9d7a326}$$
$$\texttt{8c209525d172e395ee1243d637ec4e5d46125b14570911b920a13ce3bfd}$$

$$a_{\frac{\mathbf{i}+\mathbf{j}}{2}}(Q_0) = \texttt{0x30c2adafbf1ae17fb29940d7fecac87357f40e25fc488b8e1475f960571ec63f97}$$
$$\texttt{fc25dc7bb8599bc9eb8a849033ed12c936595f41b30a8042771f896c893}$$

$$b_{\frac{\mathbf{i}+\mathbf{j}}{2}}(Q_0) = \texttt{0x46a44a940b876925af912bd0611b4b4877dd617680bef97d9b9d4c2e5c412ec206}$$
$$\texttt{f44e582749d9ae881c5f345b0093cd32759da0456c945c04fd880b355eb}$$

$$a_{\frac{1+\mathbf{k}}{2}}(P_0) = \texttt{0xbe4cfe02f20371879999c9e4d43a981f0ccd3bfd9aee73c672a79b3553061e04be}$$
$$\texttt{4c531caaad446896a8ff8f1735773d14be2465f086ebaf8e828446df9da}$$

$$b_{\frac{1+\mathbf{k}}{2}}(P_0) = \texttt{0xe42b9a2f410164c0d99f2d25fff06e2a95efe7ea1e8d77fcd187a45b43496df668}$$
$$\texttt{6cd3fe04d3b572f54d5faca3f8145d9cf9cd3d7736df50149cef1b7f72a}$$

$$a_{\frac{1+\mathbf{k}}{2}}(Q_0) = \texttt{0xbea8f8c70930891e933d02de870fc99402c6830b487aa2c756e74a3c4d5dfb9869}$$
$$\texttt{7451a39b37c2440fe791ecc666cde5c258d2c5446976573bde00df9deef}$$

$$b_{\frac{1+\mathbf{k}}{2}}(Q_0) = \texttt{0x41b301fd0dfc8e786666361b2bc567e0f332c40265118c398d5864caacf9e1fb41}$$
$$\texttt{b3ace35552bb9769570070e8ca88c2eb41db9a0f791450717d7bb920627}$$

NIST-V:

$$a_{\mathbf{i}}(P_0) = \texttt{0x5769e03941e3989403b9d6ad109a35c31a5a335d3957ad7bc3ca79ef308fc6eb7efa}$$
$$\texttt{724a347ad3a37fb4a028cf17188c7a9b180f19d0db447cd0a61bb6007ad70e3931281d}$$

58f27b207c4173fe74599f4867ed

$b_{\mathsf{i}}(P_0) =$ 0x2667f5883cc14096fbb5a4fd2581e2709f1f2b0c28e3758884e3d634c8fc74e1bb30
3a7ada0ed30be50ea61c28493978ddd7e3fa9c73b24e1729721541b48dd21c28396888
3e5013c8c0924846f7e4a6f8cd0f

$a_{\mathsf{i}}(Q_0) =$ 0x92695c791320436b2dcc1165406ff9075e1aa388472b861b19eb864f692dc5f5d323
747965a4972659f612d6588f31b2bf857b7f9677a33cf50fe64632fa69d33caf9013d8
6c994909f501a4c2fac05399d40a

$b_{\mathsf{i}}(Q_0) =$ 0xa8961fc6be1c676bfc462952ef65ca3ce5a5cca2c6a852843c358610cf7039148105
8db5cb852c5c804b5fd730e8e7738564e7f0e62f24bb832f59e449ff8528f1c6ced7e2
a70d84df83be8c018ba660b79813

$a_{\frac{\mathsf{i}+\mathsf{j}}{2}}(P_0) =$ 0x759928b8b41d5c6f920e6eb471a372c72654c241ad66cfd5ae20238c18dd7dc79a
9b3c6150773505fafff7dacba7e00e590f6dd62838e6e904065c2417ee6bec6c6c83
735b4bacce99c513fe86c8899ad796fb

$b_{\frac{\mathsf{i}+\mathsf{j}}{2}}(P_0) =$ 0x630a95245ac4379ce2d7df573c52bde423fce6da91c35c4b9bbb5e2b3ea54ab032
0dd53ccebbcfccec77a41b082cd8646f938b27c53664602d021b81291ae116d45422
6e263fc8003c89f5654ac4a2cda70da3

$a_{\frac{\mathsf{i}+\mathsf{j}}{2}}(Q_0) =$ 0xdf8abb0078b73e715744bbaf64dcc43c4a6a3ca53850a96117a706d8f7b9b86bdd
a8e2d09da648b8e0ee8abdb0be9f0050b361fad2f2046ac9fd0842597371f9c5a7d8
099fb3102253bd1b87d6ca156b8802ed

$b_{\frac{\mathsf{i}+\mathsf{j}}{2}}(Q_0) =$ 0x8a66d7474be2a3906df1914b8e5c8d38d9ab3dbe5299302a51dfdc73e722823865
64c39eaf88cafa0500082534581ff1a6f09229d7c71916fbf9a3dbe811941393937c
8ca4b45331663aec0179377665286905

$a_{\frac{1+\mathsf{k}}{2}}(P_0) =$ 0x20926b15f5cd2cc369c221e1494a8cf3d3ff5dc5b4aa94497e7c26aee7b69c37c0
ad6d23b79dec89e40c7c2c1717dbc2428d7e63f675850e8e5f96bf570d6cbd77fc67
54c963b4a13105a61e9bac89d22a47be

$b_{\frac{1+\mathsf{k}}{2}}(P_0) =$ 0xdeb2e1eec836854510bafd317662682a97f3a48ad3a0588ad9cce3589862e611cb
b13d455e96b15943135e912dd3321fb25a756321def983a33fa9adb4e26cb14306c3
498e657cb45736f8eaaca4c60eb4a2ce

$a_{\frac{1+\mathsf{k}}{2}}(Q_0) =$ 0xf0dfb117c1968ae4f03f689f40a410931eae6301f7137e2cebff2bbdb284acc881
a9ef435ea675b2cf0e79b549690b09ddb8c498b0250157587349a83bfc480ff8e03d
7d2a58fb8daef0bca9a3fe819494e9b

$b_{\frac{1+\mathsf{k}}{2}}(Q_0) =$ 0xdf6d94ea0a32d33c963dde1eb6b5730c2c00a23a4b556bb68183d951184963c83f
5292dc486213761bf383d3e8e8243dbd72819c098a7af171a06940a8f29342880398
ab369c4b5ecefa59e16453762dd5b843

Elliptic Curve Computations. The following table gives the constants used in elliptic curve computations.

We now describe the basis $\mathcal{B}_0 = (P_0, Q_0)$ of $E_0[2^f]$ for each of the three parameter sets, by providing their x and y coordinates as $\mathbb{F}_{p^2}$ elements. The algorithm used to generate this basis is given in Algorithm B.1. Here, an $\mathbb{F}_{p^2}$ element z is described by $z[0]$ and $z[1]$ such that $z = z[0] + \mathsf{i} \cdot z[1]$ where $\mathbb{F}_{p^2} = \mathbb{F}_p[\mathsf{i}]$, $\mathsf{i}^2 = -1$.

NIST-I:

$x(P_0)[0] =$ 0x2f5794ac8c12fd0aa9eecce1ef59c845e644efb1d83bfe12d9d1144badb77c74f532
d79d7af17bc4d3

TABLE 10. Global constants for lattice reduction

Constant	NIST Level			Formula
	I	III	V	
BitLenNormEqInput2DLat	163	252	334	$\frac{1}{2}\log_2(p)$
BitLenCornacchiaInput	652	1008	1336	$2\log_2(p)$
BitLenSkChall	463	641	805	$\lambda + \log_2(p) + 2\log_2$ EIBox
BitLenCommitSkChall	669	1026	1354	$2\log_2(p) + 4\log_2$ EIBox
BitLenDegMix	582	888	1180	$\log_2(p) + 2\lambda$
Z2_thresh		30		
Z2_window		63		$2 \cdot$ Z2_thresh $+ 3$
Z2_thresh_gram		29		
Z2_window_gram		61		$2 \cdot$ Z2_thresh_gram $+ 3$
Zi_thresh		27		
Zi_window		59		$2 \cdot$ Zi_thresh $+ 5$
Up_Shift		3		

Constant	NIST Level			Formula
	I	III	V	
TORSION_2POWER_BYTES	41	63	83	$\lceil f/8 \rceil$

$x(P_0)[1] =$ 0x1f5213ad2a961e68b62dd1e7a52f8a3773a68fca0fd2de384700119ccad18f953972
69898753ba602e

$x(Q_0)[0] =$ 0x17ff26f770aaa7aa48966aedb5ef8c28b0849e0672ca488a3484a008019f4bf11279
512257b1a6385b

$x(Q_0)[1] =$ 0x1da20dc6de567fd2bf17e29fa4a4c8a42bef024ed5030c5fac90d216a62228926fcb
ee6d95945b36bb

$x(P_0 - Q_0)[0] =$ 0x275c77d2683d4be786ffc592d13cc1598380dff1ea4c43a970566b6c0cc71baa89ed
7b974c9fcbe0e6

$x(P_0 - Q_0)[1] =$ 0x8bb9f6ef00d5a29d48266c7948f50f1d4ad6a02848ff466134678bc02ad9aedbb651
4fbc5bef7a427

NIST-III:

$x(P_0)[0] =$ 0x9fafe5085fcb1f13d5e487f010c8026abe233871b01f4a3587f06737f9bc686ba009
922e2d459ec8f149c4c4083604e7842a612b6fdf8180025cdeb187b4c0

$x(P_0)[1] =$ 0xc42a516ef3cf80d3e2e7a2d88faba1e46785ddce14f150ff4d204a43d47ad8d01940
b2eba9aaac28b7198e48ed9281128f5782cdd197f48cddfbffe867063d

$x(Q_0)[0] =$ 0xbce91be61859cd3ddcd3f8408657d1d43c6f2764437e66e96371e74bc4b725f0cb99
b58b09a91e872afcebb4608219f68aa3572c70ef5e6e654099bfc8aa09

$x(Q_0)[1] =$ 0xa48c9987de3810adbf0813505a561e134f31d64466875f90e21dd7c6b44eb81bed0e
58d70ebd39fd9443a7523049993bd75145d72ec8f52be58b7086fffbe8

$x(P_0 - Q_0)[0] =$ 0x2a3165c5061348beca06a8b4316633e6fb8f64fc96a2354b8fa271a9f539a4d336ee
88c0e0a5a22de6eb125da3ddc872cc0a91a5277c7aaf98ffc3d75e2d34

$x(P_0 - Q_0)[1] =$ 0xb27e86c1eff34ce8995a3b707f152e907a662ffec7e5df6ff23b36ef3329fb1ebd12
f2b2bf3459806f7905d73a8287e41c04b3936734761950daca2d3b3b4f

NIST-V:

$$x(P_0)[0] = \texttt{0xb66b93a890673a0d7e33340c53b6b986c1bbb905054419c400e5458a8655c166345a}$$
$$\texttt{bc1e4d70516e0dfcd03f7cbca2a87da6ef003f9301aa1e3a662f2db25e00f57345faa3}$$
$$\texttt{48b8c78d403891c85677b4b8d62ce}$$
$$x(P_0)[1] = \texttt{0x900d04c251411f3be86fc8bc618b7d4caefb2a31c23b1cd9f86fcfb8cb114bf6fe9e}$$
$$\texttt{19bca9fdd6e150b8fcca7eb7c7b71dceea11abff9da7dbab00e5d6e1dbe6b449796c96}$$
$$\texttt{b4f2f5a5bf4e3cb04973bc3fdb8c9}$$
$$x(Q_0)[0] = \texttt{0xf4f3311a4df85d112f2ec580f3f677f1d80c2105cf9e7dea9f8b06d3b412aa14a0e1}$$
$$\texttt{d1ae2dbf8b5d8fca950f0d355fa25cd98e4e14d58df00ada18c8453d45702ad7b8188d}$$
$$\texttt{55a8df97c0dc87782f17e9e99452c}$$
$$x(Q_0)[1] = \texttt{0x2b3e8126b1983e58e41e2ed4ef551b7f4617364b1c3f04f072d50e9d77e5ae2ecedf}$$
$$\texttt{5d924fe21e881a085764064bc175daa0308f13d6a1b760378c23aa4c912fe365ce0d56}$$
$$\texttt{73b4edf297440304f2c1415d2b502}$$
$$x(P_0 - Q_0)[0] = \texttt{0xb7a63f245737a562b92dffd7dfbb01b2b26578a58c10599d2a038a3c1368d621a7a5}$$
$$\texttt{a5114c1336c36fae4544fcc60c309f3aa7bd5ba91243c2867d9cc148d149cc30875f71}$$
$$\texttt{35401845fdf094f0c56fbcaa884df}$$
$$x(P_0 - Q_0)[1] = \texttt{0x8ea5a35b9423ca6607d04834431cb960256b15f473eacd2d1e116c8df428f29de9fb}$$
$$\texttt{db889dfbf41b4c8b8c88c5b79e2fa036ef0cc6ace710007ad97ea49709550e421ce5e1}$$
$$\texttt{5173570f007d672b77b1d4f027568}$$

CHAPTER 5

Parameters

5.1. Parameter requirements

Each parameter set consists of a prime p (the characteristic of the finite field) and a security parameter λ. All other parameters can be deduced from p, λ, and the above specification (Section 4.7). The prime p is selected under the following considerations:

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

(2) 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.

5.2. 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 and a prime p. For convenience, to make the values of f and c such that $p = c \cdot 2^f - 1$ explicit, we denote $p = c \cdot 2^f - 1$ by $p_{f.c}$.

NIST-I:

$$\begin{aligned}\lambda^{\mathrm{I}} &= 128\\ p_{324.3} &= 3 \cdot 2^{324} - 1\\ &= \texttt{0x2ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff}\\ &\quad\texttt{fffffffff}\end{aligned}$$

NIST-III:

$$\begin{aligned}\lambda^{\mathrm{III}} &= 192\\ p_{500.27} &= 27 \cdot 2^{500} - 1\\ &= \texttt{0x1affffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff}\\ &\quad\texttt{fffffffffffffffffffffffffffffffffffffffffffffffffffffff}\end{aligned}$$

NIST-V:

$$\begin{aligned}\lambda^{\mathrm{V}} &= 256\\ p_{664.17} &= 17 \cdot 2^{664} - 1\\ &= \texttt{0x10ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff}\\ &\quad\texttt{ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff}\\ &\quad\texttt{ffffffffffffffffff}\end{aligned}$$

Note. To stimulate research in the security and optimization of SQIsign, we collect in Appendix P a list of alternate parameter that may turn out to better fit NIST's security targets.

CHAPTER 6

Encodings and Data Formats

TABLE 11. Constants for encoding

Constant	NIST Level			Formula
	I	III	V	
FP_ENCODED_BYTES	41	64	84	$\lceil \log_2(p)/8 \rceil$
CHALLENGE_BYTES	16	24	32	$\lambda/8$
RESPONSE_BYTES	25	38	51	$\lceil (e_{\mathsf{rsp}} + 2)/8 \rceil$
PUBLICKEY_BYTES	83	129	169	$1 + 2 \cdot$ FP_ENCODED_BYTES
SECRETKEY_BYTES	270	417	549	PUBLICKEY_BYTES $+ 3 \cdot$ FP_ENCODED_BYTES $+ 4 \cdot$ CHALLENGE_BYTES
SIGNATURE_BYTES	200	306	406	$2 + 2 \cdot$ FP_ENCODED_BYTES $+ 4 \cdot$ RESPONSE_BYTES + CHALLENGE_BYTES

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 $p - 1$ (inclusive), in little-endian order, using FP_ENCODED_BYTES bytes.
- Elements of $\mathbb{F}_{p^2}$ are encoded by encoding the real part followed by the imaginary part, as $\mathbb{F}_p$ elements.
- Elliptic curves $y^2 = x^3 + Ax^2 + x$ are encoded by their Montgomery coefficient $A \in \mathbb{F}_{p^2}$.
- Hints for accelerating deterministic basis generation are encoded as a pair (h_A, h), where the quantity h_A is either 0 or 1 and h is a 7-bit integer. A hint (h_A, h) is then encoded as one byte with value $h_A + 2h$.
- 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.
- 2×2 integer matrices

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix}$$

  are encoded as the concatenation of the encodings of a, b, c and d, in this order.
- Inert ideals are encoded as tuples (N, x, y) where each component has size FP_ENCODED_BYTES bytes; N is the norm of the ideal and x, y are the coefficients defining the ideal as described in Chapter 3.

Public key

The public key is encoded using $2 \times$ FP_ENCODED_BYTES $+ 1$ bytes as follows:

- The first $2 \times$ FP_ENCODED_BYTES are the encoding of the public curve E_{pk} produced by SQIsign.KeyGen.
- The last byte encodes the hint obtained by running TorsionBasisToHint on E_{pk}.

Secret key

The secret key is encoded using $5 \times \texttt{FP\_ENCODED\_BYTES} + 4 \times \texttt{CHALLENGE\_BYTES} + 1$ bytes as follows:

- The first $2 \times \texttt{FP\_ENCODED\_BYTES} + 1$ bytes are the encoding of the corresponding public key, as detailed above.
- The following $3 \times \texttt{FP\_ENCODED\_BYTES}$ are the encoding of the secret ideal I_{sk} computed in SQIsign.KeyGen.
- The last $4 \times \texttt{CHALLENGE\_BYTES}$ encode the change of basis matrix from the public basis $(P_{\mathsf{pk}}, Q_{\mathsf{pk}})$ to the secret basis $(P_{\mathsf{sk}}, Q_{\mathsf{sk}})$ on E_{pk}, computed using ChangeOfBasis during SQIsign.KeyGen.

Signature

The signature is encoded using $2 \times \texttt{FP\_ENCODED\_BYTES} + 4 \times \texttt{RESPONSE\_BYTES} + \texttt{CHALLENGE\_BYTES} + 2$ bytes as follows:

- The first $2 \times \texttt{FP\_ENCODED\_BYTES}$ are the encoding of the auxiliary curve E_{aux} computed in SQIsign.Sign.
- The following $4 \times \texttt{RESPONSE\_BYTES}$ are the encoding of the matrix $\mathbf{M}_{\mathsf{chl}}$ mapping the deterministic basis $(P_{\mathsf{chl}}, Q_{\mathsf{chl}})$ into the response basis $(P_{\mathsf{rsp}}, Q_{\mathsf{rsp}})$; each entry of the matrix is represented by an integer of $\texttt{RESPONSE\_BYTES}$ bytes.
- The following $\texttt{CHALLENGE\_BYTES}$ bytes encode the challenge.
- The last two bytes encode the hints on the auxiliary curve E_{aux} and the challenge curve E_{chl} respectively, computed by TorsionBasisToHint.

CHAPTER 7

Performance Analysis

The submission package includes a reference implementation written in portable C (C11) and two additional implementations with assembly-optimized field arithmetic: one targeting the Intel Broadwell architecture (and later), and one targeting 64-bit ARM (AArch64) cores. Furthermore, assembly-optimized field arithmetic for the 32-bit ARM Cortex-M4 is provided for use with the `pqm4` framework (see Section 7.6). All implementations share the same code base of the overall SQIsign library. The library is self-contained and has no external dependencies; in particular, all arbitrary-precision integer arithmetic is implemented from scratch, and no GMP installation is required. 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, isogeny and pairing computations.
- `quaternion`: module for computations with quaternion orders and ideals, including a constant-time lattice reduction package. Its integer arithmetic is provided by the `mp` module.
- `precomp`: module with pre-computed constants for the code package.
- `gf`: implementation of the finite field arithmetic over $\mathbb{F}_p$ and $\mathbb{F}_{p^2}$. The parameter-dependent $\mathbb{F}_p$ arithmetic is automatically generated using scripts, which eases the addition of new parameter sets.
- `hd`: computations with (2,2)-isogenies in the theta model.
- `mp`: routines for multiprecision arithmetic of integers, with no dependency on external libraries such as 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, it does not run in constant-time in its entirety: the finite-field, elliptic-curve, pairing and isogeny arithmetic (modules `gf`, `ec` and `hd`) as well as the lattice reduction package in the `quaternion` module are implemented in constant time, whereas the remaining parts of the `quaternion`, `mp` and `id2iso` modules are not constant-time.

Throughout this chapter, parameter sets are identified by their targeted NIST security level (NIST-I, NIST-III and NIST-V), corresponding to the primes $p_{324.3}$, $p_{500.27}$ and $p_{664.17}$ from Section 5.2. In the submission package, the corresponding parameter sets (build targets, benchmark binaries and KAT files) are named `p324_3`, `p500_27` and `p664_17`, respectively.

7.1. Key and signature sizes

Key and signature sizes are listed in Table 12 for each security level.

TABLE 12. SQIsign key and signature sizes in bytes for each security level.

Parameter set	Public key	Secret key	Signature
NIST-I	83	270	200
NIST-III	129	417	306
NIST-V	169	549	406

7.2. Reference implementation

For benchmarking, the reference implementation is built with the CMake options `-DSQISIGN_BUILD_TYPE=ref -DCMAKE_BUILD_TYPE=Release`. As the library no longer depends on GMP, a single build configuration suffices.

7.3. Optimized implementation

The optimized implementation is the same as the reference implementation.

7.4. Intel Broadwell optimized implementation

The first additional implementation uses assembly-optimized code targeting the Intel Broadwell architecture (and later). The assembly optimizations are applied to the finite field arithmetic in the `gf` module, while the remaining code base is re-used from the reference implementation. For benchmarking, we used the following CMake options: `-DSQISIGN_BUILD_TYPE=broadwell -DCMAKE_BUILD_TYPE=Release`.

7.5. ARM64 optimized implementation

The second additional implementation uses assembly-optimized code targeting 64-bit ARM (AArch64) cores, using only baseline `armv8-a` instructions. As for the Intel Broadwell implementation, the assembly optimizations are applied to the finite field arithmetic in the `gf` module, while the remaining code base is re-used from the reference implementation. For benchmarking, we used the following CMake options: `-DSQISIGN_BUILD_TYPE=arm64 -DCMAKE_BUILD_TYPE=Release`.

7.6. ARM Cortex-M4 implementation

The full scheme (key generation, signing and verification) of the reference implementation is 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. For each parameter set, the script generates two implementation directories: a portable `ref` implementation, and an `m4f` implementation with optimized Cortex-M4 field arithmetic in place of the portable one. A GitHub repository is provided, containing a snapshot of `pqm4` at the time of the submission, incorporating SQIsign for the 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.

7.7. Performance evaluation

All builds below use the CMake `Release` configuration (which compiles with `-O2`).

Performance evaluation of both the reference and the Intel Broadwell assembly-optimized implementations was performed on an Intel Core i7-13700K (13th generation, codename Raptor Lake) x86-64 CPU with a nominal core clock of 3.4 GHz, running Ubuntu Linux 26.04 LTS. The compiler used is clang 21. Turbo Boost was disabled during benchmarking to get consistent timings, and the benchmarks were pinned to a single performance core. The results are shown in Table 13.

Performance of both the reference and the AArch64 assembly-optimized implementations was also evaluated on 64-bit ARM platforms, specifically, on an Apple M4 Pro CPU (performance core) with a nominal core clock of 4.5 GHz, running macOS 26. The compiler used is Apple clang 17. The results are shown in Table 14.

Additionally, both the reference and the AArch64 assembly-optimized implementations were benchmarked on a Raspberry Pi 5, featuring ARM Cortex-A76 cores with a nominal core clock of 2.4 GHz. The compiler used is clang 18. The results are shown in Table 15.

TABLE 13. 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			
NIST-I	43.9	125.2	18.1
NIST-III	150.5	447.3	54.7
NIST-V	277.1	835.6	106.3
Assembly-optimized implementation for Intel Broadwell or later			
NIST-I	32.2	93.9	12.1
NIST-III	102.1	320.3	31.5
NIST-V	214.5	674.7	77.6

TABLE 14. SQISIGN performance in 10^6 CPU cycles on an Apple M4 Pro CPU. Results are the median of 1,000 benchmark runs.

Parameter set	`KeyGen`	`Sign`	`Verify`
Reference implementation			
NIST-I	28.6	79.4	11.0
NIST-III	96.1	294.4	31.7
NIST-V	166.4	513.4	59.7
Assembly-optimized implementation for AArch64			
NIST-I	20.4	60.0	7.2
NIST-III	67.0	217.0	18.2
NIST-V	130.3	425.8	41.9

TABLE 15. SQISIGN performance in 10^6 CPU cycles on a Raspberry Pi 5 (ARM Cortex-A76). Results are the median of 100 benchmark runs.

Parameter set	`KeyGen`	`Sign`	`Verify`
Reference implementation			
NIST-I	118.9	343.9	52.9
NIST-III	431.4	1281.6	183.8
NIST-V	833.2	2449.1	349.9
Assembly-optimized implementation for AArch64			
NIST-I	114.4	326.8	50.1
NIST-III	345.7	1041.9	136.3
NIST-V	811.5	2431.4	337.5

Lastly, performance, stack usage and code size of the full scheme (key generation, signing and verification) were evaluated for both the `ref` and `m4f` implementations, using the `pqm4` project, on an ST Microelectronics

NUCLEO-L4R5ZI evaluation board for the 32-bit ARM Cortex-M4-based STM32L4R5ZI microcontroller, compiled with gcc version 13.2 with pqm4's preset optimization settings for speed. Results are obtained from a single run of the benchmark. The results are shown in Table 16 and Table 17.

TABLE 16. SQISIGN performance in 10^6 CPU cycles on an ARM Cortex-M4-based microcontroller, benchmarked using the pqm4 project. Results are obtained from a single run of the benchmark.

Parameter set	KeyGen	Sign	Verify
Reference implementation (ref)			
NIST-I	564.4	1854.1	229.5
NIST-III	1886.6	5929.8	668.5
NIST-V	4200.4	13 726.2	1689.5
Optimized Cortex-M4 field arithmetic (m4f)			
NIST-I	379.6	1233.3	148.8
NIST-III	1005.6	3992.5	395.4
NIST-V	2422.3	10 023.4	881.8

TABLE 17. SQISIGN stack usage per operation and code size (text + data + bss) in KB on an ARM Cortex-M4-based microcontroller, benchmarked using the pqm4 project.

Parameter set	**Stack usage (KB)**			**Code size (KB)**
	KeyGen	Sign	Verify	
Reference implementation (ref)				
NIST-I	57	99	40	170
NIST-III	76	133	63	180
NIST-V	112	181	83	191
Optimized Cortex-M4 field arithmetic (m4f)				
NIST-I	56	99	37	173
NIST-III	75	132	56	185
NIST-V	110	179	73	194

CHAPTER 8

Security Analysis

This chapter discusses the security of SQIsign. In Section 8.1, we prove that SQIsign is existentially unforgeable against chosen-message attacks (EUF-CMA) under two isogeny-based hardness assumptions. Then, in Section 8.2, we analyze the resistance of SQIsign to known attacks, and present the cost model from which we determined the concrete security parameters. In Section 8.3, we conclude with a discussion on various choices made for the parameters and how these choices have no known impact on security (a nothing-up-my-sleeve approach). In Section 8.4, we discuss the security of SQIsign in stronger adversarial models.

8.1. Security reductions

We analyze the security of SQIsign by relying on the *Fiat-Shamir with hints*[1] framework [ABD+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 [ABD+25] shows that such a 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 their main results, we refer the reader to [ABD+25] for a more thorough treatment.

The results in [ABD+25] focus on a previous version of SQIsign in which the responses may have even degree [AAA+24], whereas the current version of SQIsign works only with odd-degree responses.

To rely on the previous analysis, we modify the SQIsign Σ protocol associated with the signature from Section 2.3. In this protocol, if SampleQuaternionResponse finds no odd-degree response isogeny, the prover returns $E_{\mathsf{com}}, \perp$ instead of a valid isogeny. In this case, verification fails, but this event occurs with negligible probability $\leq 2^{-\lambda}$ (Section 9.5), so the Σ protocol is still correct. This modification is necessary to define the protocol's behavior when the original protocol aborts. The digital signature obtained from the Fiat-Shamir transform is equivalent to the one specified in Section 2.3.

Besides sampling an odd-degree response, several other subroutines in SQIsign may fail. However, these failures occur only with negligible probability, mainly due to the improvements from [BCE+25; DLS26]; thus, their impact on security is also negligible.

Definition of EUF-CMA security. The EUF-CMA security of a signature scheme $\Pi = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ is formulated as a game between an adversary $\mathcal{A}$ and a challenger.

Definition 8.1.1 (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 Π is EUF-CMA secure if every PPT adversary has negligible advantage in the security parameter λ.

Soundness. SQIsign defines a Σ protocol for the EndRing relation. It is 2-special sound, assuming the hardness of the OneEnd problem: given two accepting transcripts for the same commitment, it is possible to extract a non-scalar endomorphism.

[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 4.4.2.

Lemma 8.1.2. *The SQISIGN Σ protocol with odd-degree responses and even-degree challenge isogenies is special-sound for the relation*

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

Zero-knowledge. We now analyze the zero-knowledge property of the SQISIGN Σ protocol in the Fiat-Shamir with hints framework [ABD+25]. We refer to [ABD+25] for the formal definition of *hint distribution* ([ABD+25, Definition 3.1]), and *hint-assisted zero-knowledge* (wHVZK with hints, [ABD+25, Definition 3.2]). Informally, a Σ protocol has weak honest-verifier zero-knowledge (wHVZK) with hints if there exists a simulator $\mathcal{S}$ that, using q hints, can produce q simulated transcripts that are computationally indistinguishable from real transcripts produced by the honest prover.

For the SQISIGN security reduction, we consider two hint distributions, described in Fig. 4. Both distributions sample three isogenies that are used by the simulator to generate a transcript. The first distribution, $\mathcal{H}_E^{\mathsf{sim}}$, samples a challenge and a random commitment curve, followed by an odd-degree isogeny of degree at most D_{rsp} connecting the challenge curve and the commitment curve. Unlike [AAA+24], we consider in Line 3 the case in which no odd-degree isogeny of degree at most D_{rsp} connects the curves E_1 and E_2. In this case, the distribution still returns E_1, E_2 so that E_1 is distributed according to the stationary distribution. Note that, by Section 9.5, this happens with negligible probability $\leq 2^{-\lambda}$. The second distribution, $\mathcal{H}_E^{\mathsf{unif}}$, instead samples a challenge followed by an odd-degree 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_0 to E_1 and access to the distribution $\mathcal{H}_{E_0}^{\mathsf{unif}}$, it is possible to construct the distribution $\mathcal{H}_{E_1}^{\mathsf{unif}}$. Thus, the simulator would not need $\mathcal{H}_E^{\mathsf{unif}}$ to sample a challenge isogeny. Nevertheless, the formulation in Fig. 4 includes it to show the similarity to the $\mathcal{H}_E^{\mathsf{sim}}$ distribution.

$D_{\mathsf{Chall}}(E)$:
1: Sample $s \in \{0, \ldots, 2^\lambda - 1\}$.
2: $(P, Q), \square \leftarrow \mathsf{TorsionBasisToHint}(E, \lambda + 2, 0)$
3: $E', \square \leftarrow \mathsf{TwoIsogenyChain}(P + [s]Q, E, \lambda, \square)$ // The 2^λ-isogeny with kernel $\langle [4](P + [s]Q) \rangle$ has codomain E'
4: Return s, E'.

$\mathcal{H}_E^{\mathsf{sim}}$:
1: $s, E_1 \leftarrow D_{\mathsf{Chall}}(E)$.
2: Sample E_2 according to the stationary distribution[2] over the set of supersingular curves over $\mathbb{F}_{p^2}$.
3: If no odd-degree isogeny $\varphi_2 \colon E_1 \to E_2$ with $\deg(\varphi_2) < D_{\mathsf{rsp}}$ exists, return $h = (s, E_2, \perp)$.
4: Sample an isogeny $\varphi_2 \colon E_1 \to E_2$ uniformly among isogenies $E_1 \to E_2$ of odd degree $< D_{\mathsf{rsp}}$.
5: Write $d \leftarrow \deg(\varphi_2)$.
6: Sample an isogeny $\varphi_3 \colon E_2 \to E_3$ uniformly among the cyclic isogenies from E of degree $2^{e_{\mathsf{rsp}}} - d$.
7: Return $h = (s, \varphi_2, \varphi_3)$.

$\mathcal{H}_E^{\mathsf{unif}}$:
1: $s, E_1 \leftarrow D_{\mathsf{Chall}}(E)$.
2: Sample an integer d from a weighted distribution on the interval $[1, D_{\mathsf{rsp}}]$ in which each odd 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 $\varphi_2 \colon E_1 \to E_2$ uniformly among the (possibly non-cyclic) isogenies from E_1 of degree d.
4: Sample an isogeny $\varphi_3 \colon E_2 \to E_3$ uniformly among the cyclic isogenies from E_2 of degree $2^{e_{\mathsf{rsp}}} - d$.
5: Return $h = (s, \varphi_2, \varphi_3)$.

FIGURE 4. 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 8.1.3 (Computational hint-assisted wHVZK, analogue of Lemma 4.6 in [ABD+25])**.** *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 exist a $\mathcal{H}^{\mathsf{sim}}$-hint-assisted simulator and a PPT algorithm $\mathcal{B}$ such that*

$$\mathsf{Adv}^{\mathsf{hint\text{-}wHVZK}}_{SQISIGN, \mathcal{H}^{\mathsf{sim}}}(\mathcal{A}, q) \leq \mathsf{Adv}^{\mathsf{EndRing}_p}(\mathcal{B}) + q \cdot \Delta(\mathcal{D}_{\mathsf{com}}, S)$$

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 from the stationary distribution* S*. The term* $\Delta(\mathcal{D}_{\mathsf{com}}, S)$ *is the statistical distance between* S *and* $\mathcal{D}_{\mathsf{com}}$*, the distribution of the commitment curve in the real execution of the SQISIGN* Σ *protocol.*

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. By the definition of D_{mix} in Section 4.7, we have $p \cdot 2^{\lambda} \leq D_{\mathsf{mix}}$. A classical result [ABD+25, Proposition 2.4] gives $\Delta(\mathcal{D}_{\mathsf{com}}, S) \leq \frac{1}{2^{\lambda+1}}$.

EUF-CMA security of SQISIGN. We refer to [ABD+25] for the formal definition of a *hard relation with hints* ([ABD+25, Definition 3.3]), and we write $\mathsf{Adv}^{\mathsf{hint\text{-}rel}}_{(R,\mathcal{H})}(\mathcal{A}, q)$ for the advantage of an adversary $\mathcal{A}$ against the hard relation R, given q hints from the distribution $\mathcal{H}$. For SQISIGN, we combine the relation R_{OneEnd} with the hint distribution $\mathcal{H}^{\mathsf{sim}}$ to obtain R_{OneEnd} as a hard relation with $\mathcal{H}^{\mathsf{sim}}$ hints, and we assume the hardness of the following problem.

Problem 8.1.4 (q-sim-hint-OneEnd_p)**.** Given a supersingular curve E generated by SQIsign.KeyGen and q hints $h_1, \ldots, h_q \leftarrow \mathcal{H}^{\mathsf{sim}}_E$, find a non-scalar endomorphism of E in efficient representation.

By applying the Fiat-Shamir with hints framework, we obtain that SQISIGN is EUF-CMA secure in the Random Oracle Model (ROM) under the assumption that the above problem is hard, following the same argument as in [ABD+25, Theorem 2].

As in [ABD+25], we can consider using the second hint distribution $\mathcal{H}^{\mathsf{unif}}$. This has several benefits, as we will see, but requires us to introduce an additional problem that asks us to distinguish between the two distributions.

Problem 8.1.5. Let E be a supersingular curve generated by SQIsign.KeyGen. Distinguish between the distributions $(E, \mathcal{H}^{\mathsf{sim}}_E)$ and $(E, \mathcal{H}^{\mathsf{unif}}_E)$.

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

Problem 8.1.6 (q-unif-hint-OneEnd_p)**.** Given a supersingular curve E sampled from the stationary distribution on the set of supersingular elliptic curves over $\mathbb{F}_{p^2}$ and q hints $h_1, \ldots, h_q \leftarrow \mathcal{H}^{\mathsf{unif}}_E$, find a non-scalar endomorphism of E in efficient representation.

We can now state the EUF-CMA security of SQISIGN in Theorem 8.1.7. We remark that tighter reductions may be possible, as shown e.g., in [MJ26], but we leave this for future work.

THEOREM 8.1.7 (based on [ABD+25, Theorem 3]). *In the random oracle model, for any PPT algorithm* $\mathcal{A}$ *against the EUF-CMA security of SQISIGN, there exist expected polynomial-time algorithms* $\mathcal{B}$ *and* $\mathcal{D}$ *such that*

$$\mathsf{Adv}^{\mathsf{EUF\text{-}CMA}}_{SQISIGN}(\mathcal{A}) \leq (q+1) \cdot \left(2 \cdot \mathsf{Adv}^{\mathsf{hint\text{-}rel}}_{(R_{\mathsf{OneEnd}}, \mathcal{H}^{\mathsf{unif}})}(\mathcal{B}, s) + 2 \cdot \mathsf{Adv}^{\mathsf{hint\text{-}dist}}(\mathcal{D}, s) + 2^{-\lambda}\right) + (q+1) \cdot (s+1) \cdot \frac{1}{2^{\lambda+1}} + (q+s+1)s \cdot \frac{1}{\sqrt{p}2^{\lambda}}.$$

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 probabilities for the* s-unif-hint-OneEnd_p *problem and Problem 8.1.5, respectively.*

SKETCH OF PROOF. The theorem is a direct application of [ABD+25, Theorem 1] combined with Lemmata 8.1.2 and 8.1.3, following the same reasoning as in [ABD+25, Theorems 2 and 3]. The last two terms come from:

- the statistical distance between curves sampled from the stationary distribution and the distribution of E_{pk} and E_{com} in the real execution of the SQISIGN Σ protocol;
- the min-entropy of the SQISIGN Σ protocol, bounded using [GPS20, Theorem 1].

□

Remark 8.1.8. Theorem 8.1.7 reduction is limited to the classical random oracle model, so it does not consider adversaries that may query the random oracle in superposition. To obtain a reduction also for quantum adversaries, that means in the *quantum random oracle model*, we can, as in [ABR26, Section 5], use the *Algebraic Isogeny Model*, in which we assume we have algebraic adversaries that provide an isogeny *explanation* for any returned supersingular elliptic curve. These explanations can be used to extract a non-scalar endomorphism without any rewinding of the adversary.

The advantages of reducing the EUF-CMA security of SQIsign to the q-unif-hint-OneEnd$_p$ problem are twofold. On the one hand, the q-unif-hint-OneEnd$_p$ problem 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 by Page and Wesolowski [PW24] between the Endomorphism Ring problem and the One Endomorphism problem. Such an adaptation is formalized in [ABD+25, Section 5.3], where the authors show that the q-unif-hint-OneEnd$_p$ problem is equivalent, under either a tight quantum reduction or a looser classical reduction, to the following generalization of the Endomorphism Ring problem to the hint setting.

Problem 8.1.9 (q-unif-hint-EndRing$_p$)**.** Given a supersingular curve E sampled uniformly at random and q hints $h_1, \dots, h_q \leftarrow \mathcal{H}_E^{\mathsf{unif}}$, find four endomorphisms in efficient representation that form a basis of $\mathrm{End}(E)$.

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

8.2. Resistance to known attacks

Below, we discuss known attacks on SQIsign and present the precise cost model from which we derive our parameter choices for the various security levels.

8.2.1. Endomorphism and isogeny computation

A polynomial-time algorithm to compute the endomorphism ring of a random supersingular elliptic curve would break essentially all schemes in isogeny-based cryptography, including SQIsign [EHL+18]. It is equivalent to the problem of computing an isogeny between two given curves (a necessary task for key-recovery), and to the problem of finding one non-scalar endomorphism (supporting the soundness property), up to polynomial-time reductions that tend to be very efficient in practice. Since SQIsign keys are nearly-uniformly distributed[3] among all supersingular elliptic curves over $\mathbb{F}_{p^2}$, the security of the scheme benefits from all worst-case-to-average-case reductions between these problems [MW25]: breaking these security properties requires solving the endomorphism ring problem in the worst case.

This family of equivalent problems has been studied for at least 30 years. An algorithm with time and memory complexity $p^{1/2+o(1)}$ was proposed in 1996 [Gal99; Koh96], and an algorithm with similar time complexity but polynomial (in $\log(p)$) memory requirement was proposed in 2013 [DG16]. The latter approach has been extended further in [CS22], giving rise to *generalized Delfs–Galbraith* discussed below, and the lower-order terms in the complexity were optimized in a recent series of works [CCS22; CD26; PRSU26]. Finally, an algorithm with time and memory complexity $p^{1/3+o(1)}$ was proposed in 2026 [Wes26]: This algorithm fits into the *generalized Delfs–Galbraith* framework (which had, however, missed the crucial optimization that testing for many orientations simultaneously can be batched effectively) and permits smooth tradeoffs between the time-optimal version (using $p^{1/3+o(1)}$ time and memory) and a space-optimal version (using $p^{1/2+o(1)}$ time and $(\log p)^{O(1)}$ memory, thus meeting the complexity of the previous best known attacks). Our security estimates for SQIsign are based on this latest algorithm, including recently discovered or anticipated refinements.

8.2.1.1. Brief overview of the fastest known algorithm. The algorithm presented in [Wes26] takes as input a supersingular elliptic curve $E/\mathbb{F}_{p^2}$, and returns a non-scalar endomorphism $\alpha \in \mathrm{End}(E)$ in time $p^{1/3+o(1)}$. The main step consists in finding a separable isogeny φ from E to its Galois conjugate $E^{(p)}$ (the elliptic curve whose equation is given by the equation of E with coefficients raised to the power p; it satisfies $j(E^{(p)}) = j(E)^p$). Once such an isogeny $\varphi\colon E \to E^{(p)}$ has been found, the algorithm returns $\pi \circ \varphi\colon E \to E$ where $\pi\colon E^{(p)} \to E$ is the Frobenius isogeny.

The problem is then reduced to: given E and $E^{(p)}$, find a separable isogeny $\varphi\colon E \to E^{(p)}$. This particular case of the isogeny problem has the following guarantee: there always exists an isogeny from E to $E^{(p)}$ of degree at most $(p/2)^{1/3}$ (see [AOV26]). The algorithm proceeds by searching for such a particularly short isogeny via a *meet-in-the-middle* approach:

[3] More precisely, statistically close to the stationary distribution, see [ABD+25, Proposition 2.4] for definitions and proof.

(1) Fix a set S of positive integers.
(2) Build the table $L = \{(E', \psi) \mid \psi\colon E \to E' \text{ and } \deg(\psi) \in S\}$ of all isogenies from E of degree in S (stored as a dictionary mapping E' to ψ).
(3) For each $(E', \psi) \in L$, if $E'^{(p)}$ is the index of an entry $(E'^{(p)}, \eta) \in L$, return $\widehat{\eta}^{(p)} \circ \psi\colon E \to E^{(p)}$.

The algorithm succeeds if and only if there exists an isogeny $\varphi\colon E \to E^{(p)}$ such that $\deg(\varphi) = s_1 s_2$ for two integers $s_1, s_2 \in S$. If the algorithm fails, one can rerandomize the input E via an isogeny walk, and try again. The complexity of the algorithm is dominated by the complexity of computing the table L, and by the probability of success of each attempt. Choosing S to be a set of smooth integers smaller than $\approx p^{1/6}$ leads to an overall complexity in $p^{1/3+o(1)}$ in both time and memory.

8.2.1.2. Estimating the cost of the attack. The attack of Wesolowski [Wes26] enumerates B-smooth isogenies from a curve E and stores the j-invariants of the resulting curves in a hash table, looking for a collision with their Frobenius conjugates. Its asymptotic complexity of $p^{1/3+o(1)}$ does not directly provide a concrete estimate of the security of the SQIsign parameter sets. In practice, the actual computational cost is heavily influenced by the choice of the smoothness bound B and by the amount of memory available to the adversary. For our parameter selection, we decided to use a conservative memory-bounded cost model that gives the attacker a large but finite amount of memory while, at the same time, making optimistic assumptions about the computational cost of the attack.

A crucial observation for the concrete cost analysis of this attack is that the isogenies stored in the table need not be computed independently. Many of them share common prefixes, so the enumeration can instead be organized as a tree of variable-degree isogenies.

To give a rough idea of the batching strategy, let $\ell < B$ be a prime and suppose that $M > \ell$ curves isogenous to E by isogenies of degree less than B/ℓ have already been obtained. Their ℓ-isogenous neighbors have degree less than B and yield approximately $M\ell$ new curves. These neighbors can be computed in batches. First, one evaluates the modular polynomial $\Phi_\ell(X, Y)$ in the first variable at the M known j-invariants. Dividing the inputs into approximately M/ℓ blocks of size ℓ, each coefficient of $\Phi_\ell(X, Y)$ can be evaluated on a block using fast multipoint evaluation. Since Φ_ℓ has degree $\ell + 1$ in each variable, this step has a total cost of $\widetilde{O}(M\ell)$ field operations. For each specialized polynomial $\Phi_\ell(j(E'), Y)$, its roots give the ℓ-isogenous non-backtracking neighbors of E'. Root finding for these completely split polynomials can also be performed in quasi-linear time in their degree, up to logarithmic factors [GG13, §14.3–14.5].

Hence, approximately $M\ell$ new curves can be generated in $\widetilde{O}(M\ell)$ field operations, up to factors logarithmic in p. The amortized cost per generated j-invariant is only polylogarithmic rather than proportional to the degree of the complete isogeny.

Remark 8.2.1 (On the amortized cost of the j-invariant tree in [Wes26])**.** Our more detailed analysis of the batching strategy shows that, when the roots of the specialized modular polynomials are computed using Cantor–Zassenhaus, the amortized cost of generating each j-invariant in the tree is $\Omega\big(\log^2(B)\log^2(p)\big)$ binary operations. This estimate comes from combining fast multipoint evaluation with the cost of factoring completely split polynomials over $\mathbb{F}_{p^2}$ (cf. [GG13, §14.3–14.5]).

However, we consider the precise value of these polylogarithmic factors sensitive to further advances in computer algebra and to the exact choice of root-finding algorithm. For this reason, instead of relying on the precise asymptotic estimate, we use it only to derive a conservative lower bound on the cost of generating each j-invariant in our concrete security estimates.

Having discussed the cost of generating the j-invariants stored in the table, we now consider the time-memory tradeoffs available to the attacker: *"Going under"* is a variant of the attack that reduces the memory requirement by randomizing the starting curve until an unusually short suitable isogeny is encountered. In the parameter range considered here, our more detailed analysis gives a cost of at least $3\sqrt{p/w}$ generated j-invariants when the available memory is limited to w entries. For simplicity, we use the weaker bound $\sqrt{p/w}$ in our concrete estimates, losing only $\log_2(3) \approx 1.6$ bits while avoiding dependence on the precise smoothness model. Although we also considered the parallel collision-search algorithm of van Oorschot and Wiener [OW99], our analysis did not show a performance advantage over "going under" for the estimates considered here.

We now return to the concrete security estimates. We bound the maximum number of stored j-invariants by $w = 2^{80}, 2^{120}, 2^{160}$ entries for NIST security levels I, III, and V, respectively. The level-I value follows the memory bound historically used in the SIKE security analysis [SIK22]. For the higher security levels, the exponent is increased proportionally to the targeted classical security level, allowing proportionally larger memory budgets for adversaries targeting the higher security categories [AAC+22].

For the selected parameter sets, with $\log_2(p) = 325, 504, 668$, respectively, the estimate $\sqrt{p/w}$ corresponds to approximately $2^{122.5}$, 2^{192}, and 2^{254} generated j-invariants for NIST security levels I, III, and V. As discussed above, generating each j-invariant requires a non-negligible cost that increases with the field size. Applying the more detailed Cantor–Zassenhaus estimate $\Omega(\log^2(B)\log^2(p))$, along with the corresponding smoothness bounds, gives approximately $2^{27.3}$, $2^{29.8}$, and $2^{31.4}$ binary operations per generated j-invariant. This results in total cost exponents of approximately 149.8, 221.8, and 285.4, respectively, which exceed the corresponding NIST targets of 143, 207, and 272.

We finally discuss the effect of quantum attacks on the security classification of the selected parameter sets. A highly optimistic quantum estimate is obtained by adapting Tani's quantum claw-finding algorithm to the meet-in-the-middle structure of the attack [Tan09]. Here, one may consider two search domains of approximate sizes $|X| = p^{2/9}$ and $|Y| = p^{4/9} = |X|^2$, leading, in an ideal oracle model, to a query complexity $O(\sqrt{|Y|}) = O(p^{2/9})$. The same query bound can be viewed as precomputing the smaller side X and using Grover search over Y for an element colliding with the precomputed table.

For the selected parameter sets, this optimistic estimate corresponds to quantum query exponents of approximately 72.2, 112, and 148.4 for security levels I, III, and V, respectively. These values exceed the corresponding naive Grover query-complexity benchmarks $2^{\lambda/2}$, with $\lambda \in \{128, 192, 256\}$, namely 2^{64}, 2^{96}, and 2^{128}. Although the $O(p^{2/9})$ exponent is asymptotically smaller than the $\widetilde{O}(p^{1/4})$ complexity of the established quantum attack of Biasse, Jao, and Sankar [BJS14], the two estimates are measured in different computational models and should not be directly compared as end-to-end attack costs. The former is an optimistic query-complexity bound in an ideal oracle model, whereas the latter corresponds to a complete quantum algorithm. Together with the classical security estimates above, these comparisons support the classification of the parameter sets at NIST security levels I, III, and V [Nat26].

Remark 8.2.2 (Practical considerations of quantum attacks)**.** The $p^{2/9}$ estimate is an optimistic *query-complexity* bound in an ideal oracle model, rather than a concrete quantum implementation cost. Realizing such an attack requires efficient coherent evaluation, superposition access to the precomputed table, and practical quantum memory. Previous analyses of quantum claw finding for SIKE show that these costs can significantly reduce the apparent advantage of query-optimal algorithms [JS19]. Together with the poor parallelization of generic quantum search, this supports NIST's observation that the best practical attack may still be classical [Nat26].

Remark 8.2.3 (Dedicated parameter sets for NIST security levels II and IV)**.** The classification of dedicated parameter sets for NIST security levels II and IV is less immediate, since these levels are defined relative to the collision resistance of SHA-256 and SHA-384, respectively. A simple sufficient approach would be to require the underlying meet-in-the-middle search space to have size at least 2^{256} and 2^{384}, respectively, which heuristically corresponds to requiring $\log_2(p) \approx 3\lambda$, with $\lambda = 128$ and 192. This implies that, if dedicated parameter sets for levels II and IV are desired, primes $p_{391.3}$ and $p_{574.27}$, of 393 and 579 bits, should be chosen, respectively. Showing that smaller parameters can be justified for these two security levels would require a more refined analysis of the corresponding quantum attack costs.

8.2.2. Hardness of our security assumptions

8.2.2.1. Finding endomorphisms with hints. Forging a signature requires breaking a version of the endomorphism ring problem with hints, Problem 8.1.9, 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 efficiently sample random smooth-degree isogenies from any curve, and the two problems would thus be equivalent. In SQIsign, however, the hints also contain isogenies of non-smooth degree, which prevents a formal reduction between the two problems: this is due to the fact that sampling non-smooth degree isogenies, without additional information on the endomorphism 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, random non-smooth degree isogenies do not provide any additional information about the curve E. 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 (see Section 8.2.1).

8.2.2.2. Distinguishing uniform and simulated hints. Finally, the security proof relies on the hardness of the distinguishing game Problem 8.1.5. Our justification for its computational hardness is essentially the same as for [AAA+24, Problem 10.1.9.] in [ABD+25, Remark 5.1] (itself inspired by SQIsign2D-West [BDD+24, Sec.5.2]). Note that [AAA+24, Problem 10.1.9.] and Problem 8.1.5 are almost the same problem, with two notable differences:

(1) Problem 8.1.5 only considers odd isogenies, with a potential noticeable failure if none is present. This, however, can only happen with negligible probability, so we do not expect that a PPT adversary could exploit this in any efficient way.
(2) Problem 8.1.5 has a higher bound on the degrees of sampled isogenies φ_2. This increase is expected to make the two distributions harder to distinguish, rather than easier. Thanks to this higher bound, the number of bounded isogenies to the random target E_2 has a smaller variance, which brings the distributions of φ_2 from $\mathcal{H}_E^{\mathsf{unif}}$ and $\mathcal{H}_E^{\mathsf{sim}}$ closer.

Accounting for these two points, the arguments from [ABD+25, Remark 5.1] also apply to Problem 8.1.5.

8.2.3. 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. In SQIsign, we use a hash function HASH defined as a domain-separated $\mathsf{SHAKE256}$ truncated to λ bits of output (applied to inputs of the form $(\mathsf{pk}, E_{\mathsf{com}}, \mathsf{msg})$). This provides at least λ bits of classical security against both preimage and second-preimage attacks, and $\lambda/2$ 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 forgery attack has complexity $O(2^\lambda)$.

It is also possible to consider attacks on the Fiat–Shamir transform. While producing a pair of collisions only costs $2^{\lambda/2}$ 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 $\mathsf{HASH}(\mathsf{pk}, E_{\mathsf{com}}, \mathsf{msg}) = \mathsf{HASH}(\mathsf{pk}, E_{\mathsf{com}}, \mathsf{msg}')$, pretend to sign msg, and later deny this by claiming to have signed msg'. This attack only requires (fixed prefix) collisions, and hence it can be carried out in time $O(2^{\lambda/2})$. 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.

8.3. 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 this choice significantly simplifies certain computations. This is a standard choice in the literature which ensures 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 (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).
- 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 [MW25], making the scheme's security dependent on the problem's worst-case hardness.

8.4. Stronger security notions

8.4.1. 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-three 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 $\mathsf{chl} = \mathsf{HASH}_\lambda(\mathsf{pk}, E_{\mathsf{com}}, \mathsf{msg})$ of length λ 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 $\mathsf{chl}_{2\lambda} = \mathsf{HASH}_{2\lambda}(\mathsf{pk}, E_{\mathsf{com}}, \mathsf{msg})$ of bit length 2λ as part of the signature ensures sufficient collision resistance. During signing and verifying, only the first λ 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 λ bits, as above. This modification is similar to the way ML-DSA achieves BUFF security, and increases the signature size in SQIsign by $\lambda/8$ bytes due to the larger size of $\mathsf{chl}_{2\lambda}$.

8.4.2. Strong unforgeability

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.

8.5. Timing side-channel resistance

Our current implementation takes further steps toward mitigating potential side-channel leakage, even though these steps are not conclusive, with a particular focus on the constant-timeness of the signing and key-generation algorithms. In the following, *constant-time* always means that the execution time of the algorithm is independent of any secret used during the signing or key-generation process, rather than that the algorithm always has the same running time. This section details which subroutines from Chapter 4 are relevant for signing side-channel resistance, which of them already come with a constant-time implementation, and which require additional research effort. We stress again that the implementations provided in this submission do not run overall in constant-time, but only some restricted portions.

Established constant-time subroutines. From [AAA+24], finite-field, elliptic-curve, and pairing-arithmetic subroutines are already constant-time.

We also note that one-dimensional 2-isogeny evaluation is already constant-time; however, SQIsign currently does not use one-dimensional isogenies in any secret-dependent way.

We provide a constant-time implementation for two-dimensional $(2, 2)$-isogeny evaluations, as described in [DMPR24; Dup25]. This consists of formulas for both gluing and general steps.

Quaternions and integers. In this submission, we rely on [Ler26; LS26], which provide improved constant-time friendly subroutines for quaternion arithmetic (see Section 4.3). These procedures provide better control over the size of the integers involved, making the procedures amenable to a fixed-size and fixed-point integer-arithmetic implementation. This implementation also comes with its own integer arithmetic module (Section 4.1), replacing GMP[4] used in the previous submissions [AAA+24; CSD+23]. This not only removes an external dependency but also lays the foundation for a future side-channel-resistant and constant-time implementation. However, in the current state of research, generic constant-time implementations of the subroutines from Section 4.1 would incur a wide range of performance penalties. Thus, we encourage additional research on these topics, not only to improve their efficiency but also to tailor their use to the specific needs of SQIsign.

Lattice reductions. The relevance of lattice reduction for the constant-timeness of the signing algorithm has been discussed and partially addressed in [HKK+25]. In our new implementation, we are finally able to provide efficient constant-time algorithms (Section 4.3.3) to compute reduced bases for:

- lattices of dimension 2 over the integers, as a subroutine of Cornacchia and IdealToIsogeny,
- quaternion ideals, that are considered as lattices of dimension 4 over the integers,
- quaternion $\mathcal{O}_0$-left ideals, that are considered as lattices of dimension 2 over the Gaussian integers.

All these algorithms constitute a novel addition for the submission with respect to the previous one [AAA+24].

Ideal-to-Isogeny. The implemented IdealToIsogeny algorithm is not overall constant-time, and making it so would require additional research effort. The algorithm requires performing several loops that could potentially leak information about the secret input ideal. The report [DLS26] provides a detailed and rigorous analysis of the algorithm's failure probability, providing bounds on the number of loop iterations. While, in principle, these bounds could be used to achieve constant-timeness using dummy iterations, performance would be significantly—and potentially unnecessarily—affected. We hope that future research will find better solutions and refer to [DLS26] for a more detailed discussion.

Overall signing and key generation procedure. The most central subroutine for the constant-timeness of SQIsign is IdealToIsogeny, as it is used with secret-dependent inputs both in SQIsign.KeyGen and SQIsign.Sign. In particular, in the latter, it is used for generating the commitment and the higher-dimensional representation of the response isogeny.

Besides IdealToIsogeny, SQIsign.Sign also performs other secret-dependent operations. Most are already implemented in constant-time, mainly computing products, intersections, and reduced bases of secret quaternion ideals, but overall we still lack a full provable constant-time implementation, for example because of potential secret-dependent variation in the sampling of a random element of odd or prime norm and of a random auxiliary isogeny ideal.

[4] https://gmplib.org/

CHAPTER 9

Heuristics and Failure Analysis

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 or to point toward reference material when it exists.

9.1. Heuristics on lattices and ideals

The lattice reduction algorithms of Section 4.3.3 are applied to a small number of families of lattices, all of them coming from quaternion ideals: the rank-2 $\mathbb{Z}$-lattices in HNF shape reduced by LatReduction2DZ and SumOfSquares, the rank-2 $\mathbb{Z}[\mathbf{i}]$-lattices of inert ideals reduced by LehmerGLZi, and the rank-4 $\mathbb{Z}$-lattices, given as the reversed dual of a MulIdeal, reduced by DualLatReduction4DZ. Given correct bounds on their inputs and sufficiently many iterations, each of these algorithms satisfies their claims provably, even for worst-case instances. Experimentally, however, none of them shows any structure that a reduction algorithm could see, which we record as follows. Recall that $\lambda_i(\mathcal{L})$ is the *squared* i-th minimum of the lattice $\mathcal{L}$.

Heuristic 9.1.1. Each of the lattices reduced in Section 4.3.3 behaves like a uniformly random integral lattice of the same rank and determinant: over $\mathbb{Z}$ for the lattices of rank 2 in HNF shape and for the rank-4 duals of a MulIdeal, and over $\mathbb{Z}[\mathbf{i}]$ for the rank-2 lattices that are inert $\mathcal{O}_0$-ideals. In particular, in the two rank-2 cases, for every $t > 0$,

$$\Pr\left[\lambda_1(\mathcal{L}) \le 2^{-t} \det \mathcal{L}\right] \le 2^{-t} \quad \text{over } \mathbb{Z}, \qquad \Pr\left[\lambda_1(\mathcal{L}) \le 2^{-t} \det \mathcal{L}\right] \le 2^{-2t} \quad \text{over } \mathbb{Z}[\mathbf{i}],$$

and in the rank-4 case the ratios $\lambda_{i+1}(\mathcal{L})/\lambda_i(\mathcal{L})$ of consecutive minima obey tail bounds of the same shape.

Heuristic 9.1.1 is what makes the probabilistic statements of Section 4.3.3 precise. An unusually small λ_1, or equivalently an unusually large gap λ_2/λ_1, is exactly the situation in which the exact Minkowski conditions can fail to hold, so the tail bounds above bound the probability that the output is not exactly Minkowski-reduced. They also account for the canonicity of the output: two minima of size about X are absolute norms of lattice vectors, hence integers of that size, and coincide only with probability of order $1/X$. For the sizes at hand this is far below $2^{-\lambda}$, so the minima are distinct and the reduced basis is unique up to the sign rule. The same heuristic is what allows the reference implementation to replace the worst-case bounds by an average-case analysis, and to replace the exact rational arithmetic by a fixed-point representation which retains at least $2^{-\lambda}$ relative precision throughout the algorithm. Together, these allow constant-time implementations of the algorithms to preserve reasonable efficiency.

The remaining heuristic concerns the number K of tours of BKZ2-EvenOdd in DualLatReduction4DZ, for which no tight worst-case bound is available.

Heuristic 9.1.2. Let $\mathcal{L}$ be a rank-4 lattice as in Heuristic 9.1.1, whose diagonal profile spans γ bits. Then $K = \lfloor \log_2 \gamma \rfloor + 4$ tours of BKZ2-EvenOdd, each with $M = \lfloor \gamma/(\texttt{Z2\_thresh\_gram} - 1) \rfloor + 1$ iterations per block reduction, output a $\mathtt{BKZ}_2$-reduced basis of $\mathcal{L}$, except with probability $2^{-\lambda}$.

When the minima are close to balanced, a bound of $\log_2 \gamma + O(1)$ could be proven by adapting the techniques of [HPS11; Vil92] to this specific 4-dimensional setting. However, under the tail bounds of Heuristic 9.1.1 one can only prove statements of the shape $\log_2 \gamma + O(\lambda)$, but with a large constant in the $O(\cdot)$. In practice, the number of tours needed beyond $\log_2 \gamma$ is very small, leading us to state the above heuristic. Two further remarks make Heuristic 9.1.2 less demanding than it looks: the Minkowski step that follows does not need a perfectly $\mathtt{BKZ}_2$-reduced basis to start from, which leaves some slack in K; and the failure probability $2^{-\lambda}$ we claim is the combination of that slack with the fixed-point precision used in the reference implementation.

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 4.5 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 or 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 a 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 ThetaChangeOfCoordinates 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 computation 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 an $\mathbb{F}_{p^2}$ 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. Quaternion algorithms

There are two sub-algorithms that can fail in the quaternion module:

(1) RepresentInteger
(2) EquivalentCoprimeIdeal

These failures can occur either during key generation or during the commitment and response phases of signing. For both these algorithms, the failure probability is related to the following heuristics.

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] for instance, 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. RepresentInteger

RepresentInteger is called in RandomIdealGivenNorm with input $M = \mathsf{RICofactor}N$ where RICofactor is an implicit parameter of the scheme (chosen specifically to ensure the negligible failure probability of RepresentInteger) and N is the input. In SQIsign, the value N will be the degree $2^{e_{\mathsf{rsp}}} - q_{\mathsf{rsp}}$ of the auxiliary isogeny. As q_{rsp} is chosen to be smaller than $2^{e_{\mathsf{rsp}}-1}$, M is contained in the interval $[2^{e_{\mathsf{rsp}}-1}\mathsf{RICofactor}, 2^{e_{\mathsf{rsp}}}\mathsf{RICofactor}]$.

RepresentInteger calls Cornacchia to solve an equation of the form $x^2+y^2 = M'$ for $M' = M - p(z^2+w^2)$, where z, w simply need to be chosen to ensure that $M' > 0$. When this is the case, the call to Cornacchia succeeds if and only if M' is prime and equal to $1 \mod 4$.

Under Heuristic 9.3.1, M' should behave like a random integer smaller than M and thus we expect that it will be prime with probability $\approx 1/\log(M)$ and that it will be equal to $1 \mod 4$ with probability $1/2$.

Thus, trying random pairs z, w such that $M' > 0$ the failure probability after n attempts is smaller than

$$P(n, M) = (1 - \frac{1}{2\log(M)})^n$$

As there are roughly M/p distinct pairs z, w for which $M' > 0$, we see that an upper-bound on the failure probability of RepresentInteger is then $P(2^{e_{\mathsf{rsp}}}\mathsf{RICofactor}/p, 2^{e_{\mathsf{rsp}}}\mathsf{RICofactor})$ and so RICofactor is chosen as the smallest prime satisfying $P(2^{e_{\mathsf{rsp}}}\mathsf{RICofactor}/p, 2^{e_{\mathsf{rsp}}}\mathsf{RICofactor}) \leq 2^{-\lambda}$

The analysis of EquivalentCoprimeIdeal is similar. In this algorithm, on input some ideal I, we search for an equivalent ideal $J \sim I$ whose norm satisfies some coprimality constraint. The worst case will be when we require this norm to be prime.

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 $[-\mathsf{EIBox}, \mathsf{EIBox}]^4$. As the quadratif form is homogeneous, and there are some symetries du to the presence of $\mathbf{i}$ in $\mathcal{O}_0$, we're gonna lower-bound the number of possible candidates by $(\mathsf{EIBox}+1)^4$.

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

$$q_I(c_1, c_2, c_3, c_4) \approx \frac{8p}{\pi^2} \cdot \mathsf{EIBox}^2$$

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

$$\left(1 - \frac{1}{2\log(p)}\right)^{(\mathsf{EIBox}+1)^4} < 2^{-\lambda}.$$

9.3.2. Coprimality defect

We also have several algorithms in the quaternion module that require elements of coprime norm (ideals or quaternion elements). This is usually ensured by selecting an element of prime norm.

In particular, Intersect and IdealMultiplication need to take two ideals of coprime norm as input. RandomIdealGivenNorm takes as input an integer N and a quaternion element of norm coprime to N. In the signature process this implies the following constraints:

(1) The commitment degree $\mathsf{IdealNorm}(I_{\mathsf{com}})$ must be coprime to $\mathsf{IdealNorm}(I_{\mathsf{sk,chl}})$ in SampleQuaternionResponse (for IdealMultiplication)
(2) The auxiliary degree $2^{e_{\mathsf{rsp}}} - q_{\mathsf{rsp}}$ must be coprime to $\mathsf{IdealNorm}(I_{\mathsf{sk}})$ and $\mathsf{IdealNorm}(I_{\mathsf{sk,chl}})$ (for Intersect and RandomIdealGivenNorm)

The ideals I_{sk} and $I_{\mathsf{sk,chl}}$ are the outputs of calls to EquivalentCoprimeIdeal with prime norm, while I_{com} is an output of a call to EquivalentCoprimeIdeal with odd degree. Most of the values will be around $\sqrt{p}$.

Finally q_{rsp} is an output of SampleQuaternionResponse which must be smaller than $2^{e_{\mathsf{rsp}}-1}$ which implies that $2^{e_{\mathsf{rsp}}} - q_{\mathsf{rsp}} > 2^{e_{\mathsf{rsp}}-1} > \sqrt{p}$. All the numbers we consider are thus again represented by quadratic forms. Under Heuristic 9.3.1, they behave like random numbers of the same size, and thus the probability that the coprimality condition is not satisfied will be smaller than $2^{-\lambda}$ for each choice of parameters.

9.4. Ideal-to-isogeny translation

In the ideal-to-isogeny translation, some failures can occur in the execution of IdealToIsogenyNormEquation. The version of IdealToIsogenyNormEquation specified here was introduced in [DLS26]. For $p = c2^f - 1$, consider

an invocation of IdealToIsogenyNormEquation with exponent $f-2$ on a uniformly random ideal class. Under the plausible heuristics (see [DLS26, Appendix C]), the failure probability is bounded above by

$$\frac{1}{\sqrt{p}}\left(3(\log p+8)+12c(2c+\frac{1}{2})+\frac{6}{\pi}c\log(4\sqrt{c})\log p\right)+p^{-1}. \tag{16}$$

We apply this formula for our chosen parameters and get the failure probabilities reported in Table 18.

TABLE 18. Failure probability of IdealToIsogenyNormEquation across SQIsign parameters

NIST Level	$\log(p)$	c	Failure Probability
I	326	3	2^{-151}
III	504	27	2^{-236}
V	668	17	2^{-318}

9.5. Existence of odd degree responses

The current version of SQISIGN works with responses of odd degree. Thus, the condition $D_{\mathsf{rsp}} \geq 2\sqrt{2p}/\pi$, used in previous versions of SQISIGN, is not sufficient to guarantee the existence of a response (see [DLRW24, Lemma 12]). If, after computing $E_{\mathsf{com}}, E_{\mathsf{chl}}$, there is no odd degree isogeny in $\mathrm{Hom}(E_{\mathsf{com}}, E_{\mathsf{chl}})$ of degree smaller than D_{rsp}, the signature procedure will fail. We set D_{rsp} so that this happens with probability negligible in λ. To do so, we rely on the following experimentally verified heuristic on the tail distribution of the third minimum of the response lattices.

Heuristic 9.5.1. There exists a constant $C > 0$ such that for any supersingular elliptic curve E_{pk}:

$$\Pr\left[\lambda_3(\mathrm{Hom}(E_{\mathsf{com}}, E_{\mathsf{chl}})) \geq B\right] \leq C\frac{p^2}{B^4}, \tag{17}$$

where the probability is taken over:

- $E_{\mathsf{com}} \leftarrow \mathrm{cls}(I_{\mathsf{com}}) \star E_0$ with $\mathrm{cls}(I_{\mathsf{com}})$ random $\mathcal{O}_0$-ideal class;
- E_{chl} the codomain of a cyclic isogeny $\varphi\colon E_{\mathsf{pk}} \to E_{\mathsf{chl}}$ with $\deg(\varphi) = 2^\lambda$ sampled from $D_{\mathsf{Chall}}(E_{\mathsf{pk}})$;

and $\lambda_3(\Lambda)$ is the third minimum of the lattice Λ with respect to the quadratic form induced by $\deg$.

Heuristic 9.5.1 is supported by substantial experimental analysis, which estimates $C \approx 0.1$. As a consequence of Heuristic 9.5.1, we have that the procedure SampleQuaternionResponse, as a subroutine of Algorithm 3.23, returns a valid response with probability at least $1-2^{-\lambda}$ when

$$D_{\mathsf{rsp}} \geq 2\sqrt{p}\cdot 2^{\frac{\lambda}{4}}\cdot C^{\frac{1}{4}}. \tag{18}$$

This is a consequence of the following lemma.

Lemma 9.5.2. *Let E, E' be two supersingular elliptic curves over $\mathbb{F}_{p^2}$ and let $\Lambda = \mathrm{Hom}(E, E')$ be the lattice of homomorphisms from E to E'. Let $\lambda_3(\Lambda)$ be the third minimum of the lattice Λ with respect to the quadratic form induced by $\deg$. Then, there is always an odd degree isogeny $\varphi\colon E \to E'$ with $\deg(\varphi) \leq 2\lambda_3$.*

The lemma can be proved by considering the homomorphic surjection $\mathrm{Hom}(E, E') \to (\mathbb{Z}/2\mathbb{Z})^{2\times 2}$ from [BFLS26, Lemma 5] and checking that all 3-dimensional subspaces contain at least one invertible matrix.

However, the lemma does not guarantee that the odd degree isogeny can be found efficiently by sampling random lattice elements as in RandomBoundedElement. For that, we need that a constant ratio of the elements of $\mathrm{Hom}(E, E')$ of norm bounded by D_{rsp} have odd degree. This is implied by the following natural heuristic:

Heuristic 9.5.3. The image in $(\mathbb{Z}/2\mathbb{Z})^{2\times 2}$ of the intersection L of $\mathrm{Hom}(E_{\mathsf{com}}, E_{\mathsf{chl}})$ (defined in Heuristic 9.5.1) with a ball of radius B is uniformly random in the linear space spanned by the elements of L.

With Heuristic 9.5.3, we can compute a lower bound for the probability of each iteration in RandomBoundedElement giving a valid response. We first consider the case in which all the elements in the reduced basis have degree smaller than D_{rsp}. We can observe that:

- The image of the natural map $\mathrm{Hom}(E, E') \to (\mathbb{Z}/2\mathbb{Z})^{2\times 2}$ contains all the 16 possible matrices, and of these there are 6 invertible ones. Thus, if we sample a random linear combination of the four basis elements, we have a probability of $6/16 = 3/8$ to get an odd degree isogeny.
- RandomBoundedElement samples a linear combination of the reduced basis of the lattice $L = \mathrm{Hom}(E, E')$ from a coefficients box, with bounds determined from the dual lattice L^* minima. It accepts only if the element has degree smaller than D_{rsp} i.e., if it is in the Euclidean ball of radius D_{rsp}. With the basis being (reversed) dual-Minkowski reduced and thus hitting all dual minima, this happens with probability $\approx \frac{\pi^2}{2^5}\frac{\det(L^*)}{\sqrt{\prod \lambda_i(L^*)}}$, given by the ratio between the 4-dimensional sphere and parallelogram volumes. By Minkowski's second theorem in Hermite-refined form, we can lower-bound $\det(L^*) \geq \gamma_4^{-2}\sqrt{\prod \lambda_i(L^*)}$, where $\gamma_4 = \sqrt{2}$ is Hermite's constant in dimension 4. Consequently the probability is lower-bounded by $\frac{\pi^2}{64}$. This argument assumes that all sampling bounds β_i in RandomBoundedElement are sufficiently large. If e.g., $\beta_1 = 0$, then we can consider the same argument on the 3-dimensional lattice spanned by the first 3 basis vectors, giving a higher probability which we thus ignore.

Putting everything together, the probability of finding an odd degree isogeny in one iteration of RandomBoundedElement is

$$\approx p_{4,o} = \frac{3}{8} \cdot \frac{\pi^2}{2^6} \approx 0.05783 \,. \tag{19}$$

The same reasoning extends to all cases, in which only some of the basis elements have degree smaller than D_{rsp}. Thus, in all such cases, the probability of finding an odd degree isogeny in one iteration of RandomBoundedElement is always larger than 0.05783. Thus, to ensure a negligible failure probability in SampleQuaternionResponse, we can set the number of iterations in RandomBoundedElement to be

$$\mathsf{BoundIterationRBE} = -\lambda/\log_2(1 - p_{4,o}) \approx \lambda \cdot 11.636 \,. \tag{20}$$

Bibliography

[AAA+24] Marius A. Aardal, Gora Adj, Diego F. Aranha, Andrea Basso, Isaac Andrés Canales Martínez, Jorge Chávez-Saab, Maria Corte-Real Santos, Pierrick Dartois, Luca De Feo, Max Duparc, Jonathan Komada Eriksen, Tako Boris Fouotsa, Décio Luiz Gazzoni Filho, Basil Hess, David Kohel, Antonin Leroux, Patrick Longa, Luciano Maino, Michael Meyer, Kohei Nakagawa, Hiroshi Onuki, Lorenz Panny, Sikhar Patranabis, Christophe Petit, Giacomo Pope, Krijn Reijnders, Damien Robert, Francisco Rodríguez Henríquez, Sina Schaeffler, and Benjamin Wesolowski. *SQIsign*. Tech. rep. available at https://csrc.nist.gov/Projects/pqc-dig-sig/round-2-additional-signatures. National Institute of Standards and Technology, 2024 (cit. on pp. 103, 104, 109–111).

[AAC+22] Gorjan Alagic, Daniel Apon, David Cooper, Quynh Dang, Thinh Dang, John Kelsey, Jacob Lichtinger, Yi-Kai Liu, Carl Miller, Dustin Moody, Rene Peralta, Ray Perlner, Angela Robinson, and Daniel Smith-Tone. *Status Report on the Third Round of the NIST Post-Quantum Cryptography Standardization Process*. Tech. rep. NISTIR 8413-upd1. National Institute of Standards and Technology, July 2022. DOI: 10.6028/NIST.IR.8413-upd1 (cit. on p. 107).

[ABD+25] Marius A. Aardal, Andrea Basso, Luca De Feo, Sikhar Patranabis, and Benjamin Wesolowski. "A Complete Security Proof of SQIsign". In: *CRYPTO 2025, Part VI*. Ed. by Yael Tauman Kalai and Seny F. Kamara. Vol. 16005. LNCS. Springer, Cham, Aug. 2025, pp. 190–222. DOI: 10.1007/978-3-032-01887-8_7 (cit. on pp. 103–106, 109).

[ABR26] Marius A. Aardal, Andrea Basso, and Doreen Riepel. "The Algebraic Isogeny Model: A General Model with Applications to SQIsign and Key Exchanges". In: *EUROCRYPT 2026, Part IV*. Ed. by Joan Daemen and Emmanuel Thomé. Vol. 16545. LNCS. Springer, Cham, May 2026, pp. 604–635. DOI: 10.1007/978-3-032-25327-9_21 (cit. on p. 105).

[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: *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. 110).

[AOV26] Yves Aubry, Roger Oyono, and Christelle Vincent. *Minimal degree of an isogeny between a supersingular elliptic curve and its conjugate*. 2026. arXiv: 2607.14624 [math.NT]. URL: https://arxiv.org/abs/2607.14624 (cit. on p. 106).

[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". In: *PKC 2025, Part III*. Ed. by Tibor Jager and Jiaxin Pan. Vol. 15676. LNCS. Springer, Cham, May 2025, pp. 300–332. DOI: 10.1007/978-3-031-91826-1_10 (cit. on p. 108).

[BCE+25] Giacomo Borin, Maria Corte-Real Santos, Jonathan Komada Eriksen, Riccardo Invernizzi, Marzio Mula, Sina Schaeffler, and Frederik Vercauteren. "Qlapoti: Simple and Efficient Translation of Quaternion Ideals to Isogenies". In: *ASIACRYPT 2025, Part IV*. Ed. by Goichiro Hanaoka and Bo-Yin Yang. Vol. 16248. LNCS. Springer, Singapore, Dec. 2025, pp. 174–205. DOI: 10.1007/978-981-95-5113-2_6 (cit. on pp. 7, 82, 103).

[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, 109).

[BFLS26] Giacomo Borin, Luca De Feo, Guido Lido, and Sina Schaeffler. "The SQInstructor: a guide to SQIsign and the deuring correspondence with level structures". In: *Annual International Cryptology Conference*. Springer. 2026, pp. 537–569. DOI: 10.1007/978-3-032-35398-6_18 (cit. on p. 115).

[BJS14] Jean-François Biasse, David Jao, and Anirudh Sankar. "A Quantum Algorithm for Computing Isogenies between Supersingular Elliptic Curves". In: *INDOCRYPT 2014*. Ed. by Willi Meier and Debdeep Mukhopadhyay. Vol. 8885. LNCS. Springer, Cham, Dec. 2014, pp. 428–442. DOI: 10.1007/978-3-319-13039-2_25 (cit. on p. 108).

[BZ98] Christoph Burnikel and Joachim Ziegler. *Fast Recursive Division*. Tech. rep. MPI-I-98-1-022. Max-Planck-Institut für Informatik, 1998 (cit. on p. 26).

[CCS22] Maria Corte-Real Santos, Craig Costello, and Jia Shi. "Accelerating the Delfs-Galbraith Algorithm with Fast Subfield Root Detection". In: *CRYPTO 2022, Part III*. Ed. by Yevgeniy Dodis and Thomas Shrimpton. Vol. 13509. LNCS. Springer, Cham, Aug. 2022, pp. 285–314. DOI: 10.1007/978-3-031-15982-4_10 (cit. on p. 106).

[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 pp. 3, 13).

[CD26] Jesús-Javier Chi-Domínguez. "Improved Subfield Curve Search For Specific Field Characteristics". In: *IACR Communications in Cryptology* 3.2 (Aug. 3, 2026). ISSN: 3006-5496. DOI: 10.62056/ah5womja5 (cit. on p. 106).

[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. 110).

[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. 33).

[CS22] Mathilde Chenu and Benjamin Smith. "Higher-degree supersingular group actions". In: *Transactions on Mathematical Cryptology* 1.2 (Mar. 2022), pp. 85–101. URL: https://journals.flvc.org/mathcryptology/article/view/130604 (cit. on p. 106).

[CSD+23] Jorge Chavez-Saab, Maria Corte-Real Santos, Luca De Feo, Jonathan Komada Eriksen, Basil Hess, David Kohel, Antonin Leroux, Patrick Longa, Michael Meyer, Lorenz Panny, Sikhar Patranabis, Christophe Petit, Francisco Rodríguez Henríquez, Sina Schaeffler, and Benjamin Wesolowski. *SQIsign*. Tech. rep. available at https://csrc.nist.gov/Projects/pqc-dig-sig/round-1-additional-signatures. National Institute of Standards and Technology, 2023 (cit. on p. 111).

[Dar25] Pierrick Dartois. "Fast computation of higher dimensional isogenies for cryptographic applications". PhD thesis. Université de Bordeaux, France, July 2025. URL: https://theses.fr/s364682 (cit. on pp. 7, 71, 82, 122–125).

[DG16] Christina Delfs and Steven D. Galbraith. "Computing isogenies between supersingular elliptic curves over $\mathbb{F}_p$". In: *DCC* 78.2 (2016), pp. 425–440. DOI: 10.1007/s10623-014-0010-1 (cit. on p. 106).

[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 p. 3).

[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. 114, 115).

[DLS26] Max Duparc, Antonin Leroux, and Sina Schaeffler. *Qlapoty: Improved analysis and efficiency for quaternionic ideal to isogeny transformation*. Cryptology ePrint Archive, Paper 2026/1700. 2026. URL: https://eprint.iacr.org/2026/1700 (cit. on pp. 7, 82, 83, 103, 111, 114, 115).

[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. 65, 69, 110, 122).

[Dup25] Max Duparc. "Superglue: Fast Formulae for (2,2)-Gluing Isogenies". In: *ASIACRYPT 2025, Part IV*. Ed. by Goichiro Hanaoka and Bo-Yin Yang. Vol. 16248. LNCS. Springer, Singapore, Dec. 2025, pp. 372–400. DOI: 10.1007/978-981-95-5113-2_12 (cit. on pp. 7, 70, 71, 110, 123, 124).

[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, 8, 106).

[ESWvW26] Thomas Espitau, Sina Schaeffler, Alexandre Wallet, and Wessel van Woerden. *Lattice reduction algorithms for SQIsign*. to appear. 2026 (cit. on pp. 35, 36, 45).

[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. 8).

[Gal99] Steven D. Galbraith. "Constructing Isogenies between Elliptic Curves Over Finite Fields". In: *LMS Journal of Computation and Mathematics* 2 (1999), pp. 118–138. DOI: 10.1112/S1461157000000097. URL: https://www.math.auckland.ac.nz/~sgal018/iso.pdf (cit. on p. 106).

[GG13] Joachim von zur Gathen and Jürgen Gerhard. *Modern Computer Algebra*. 3rd ed. Cambridge: Cambridge University Press, 2013 (cit. on p. 107).

[GPS20] Steven D. Galbraith, Christophe Petit, and Javier Silva. "Identification Protocols and Signature Schemes Based on Supersingular Isogeny Problems". In: *Journal of Cryptology* 33.1 (Jan. 2020), pp. 130–175. DOI: 10.1007/s00145-019-09316-0 (cit. on p. 105).

[HKK+25] Ottó Hanyecz, Alexander Karenin, Elena Kirshanova, Péter Kutas, and Sina Schaeffler. "Constant time lattice reduction in dimension 4 with application to SQIsign". In: *IACR TCHES* 2025.2 (2025), pp. 511–534. DOI: 10.46586/tches.v2025.i2.511-534 (cit. on pp. 41, 111).

[HPS11] Guillaume Hanrot, Xavier Pujol, and Damien Stehlé. "Analyzing Blockwise Lattice Algorithms Using Dynamical Systems". In: *CRYPTO 2011*. Ed. by Phillip Rogaway. Vol. 6841. LNCS. Springer, Berlin, Heidelberg, Aug. 2011, pp. 447–464. DOI: 10.1007/978-3-642-22792-9_25 (cit. on p. 112).

[JD11] David Jao and Luca De Feo. "Towards Quantum-Resistant Cryptosystems from Supersingular Elliptic Curve Isogenies". In: *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 p. 3).

[JL17] Simon Josefsson and Ilari Liusvaara. *Edwards-Curve Digital Signature Algorithm (EdDSA)*. RFC 8032. Jan. 2017. DOI: 10.17487/RFC8032. URL: https://www.rfc-editor.org/info/rfc8032 (cit. on p. 12).

[JS19] Samuel Jaques and John M. Schanck. "Quantum Cryptanalysis in the RAM Model: Claw-Finding Attacks on SIKE". In: *CRYPTO 2019, Part I*. Ed. by Alexandra Boldyreva and Daniele Micciancio. Vol. 11692. LNCS. Springer, Cham, Aug. 2019, pp. 32–61. DOI: 10.1007/978-3-030-26948-7_2 (cit. on p. 108).

[Kan97] Ernst Kani. "The number of curves of genus two with elliptic differentials". In: *Journal für die reine und angewandte Mathematik* 1997.485 (1997), pp. 93–122. DOI: doi:10.1515/crll.1997.485.93 (cit. on pp. 68, 82).

[KLKL26] Won Kim, Jeonghwan Lee, Hyeonhak Kim, and Changmin Lee. "SQIsign with Fixed-Precision Integer Arithmetic". In: *PKC 2026, Part III*. Ed. by Shi Bai and Edoardo Persichetti. Vol. 16553. LNCS. Springer, Cham, May 2026, pp. 3–30. DOI: 10.1007/978-3-032-26737-5_1 (cit. on p. 7).

[Koh96] David Kohel. "Endomorphism rings of elliptic curves over finite fields". PhD thesis. University of California, Berkeley, 1996. ISBN: 978-0591-32123-4. URL: https://www.i2m.univ-amu.fr/perso/david.kohel/pub/thesis.pdf (cit. on p. 106).

[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. 100).

[Leh38] D. H. Lehmer. "Euclid's Algorithm for Large Numbers". In: *The American Mathematical Monthly* 45.4 (1938), pp. 227–233 (cit. on p. 26).

[Ler26] Antonin Leroux. "Efficient quaternion algorithms for the deuring correspondence, and application to the evaluation of modular polynomials". In: *ANTS XVII* (2026) (cit. on pp. 7, 45, 111).

[LS26] Antonin Leroux and Sina Schaeffler. *New algorithms for quaternion ideals in SQIsign*. to appear. 2026 (cit. on pp. 7, 45, 48, 111).

[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. 8).

[MJ26] Maher Mamah and David Jao. *A Note on the Security Proof of SQIsign*. Cryptology ePrint Archive, Paper 2026/1775. 2026. URL: https://eprint.iacr.org/2026/1775 (cit. on p. 105).

[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 pp. 3, 13, 68, 82).

[Mum66] David Mumford. "On the equations defining abelian varieties 1". In: *Inventiones mathematicae* 1.4 (1966), pp. 287–354. DOI: 10.1007/BF01389737 (cit. on p. 65).

[MW25] Arthur Herlédan Le Merdy and Benjamin Wesolowski. "Unconditional Foundations for Supersingular Isogeny-Based Cryptography". In: *TCC 2025, Part III*. Ed. by Benny Applebaum and Huijia (Rachel) Lin. Vol. 16270. LNCS. Springer, Cham, Dec. 2025, pp. 266–297. DOI: 10.1007/978-3-032-12296-4_9 (cit. on pp. 106, 110).

[Nat26] National Institute of Standards and Technology. *Post-Quantum Cryptography: Frequently Asked Questions*. https://csrc.nist.gov/Projects/Post-Quantum-Cryptography/faqs. Computer Security Resource Center. 2026 (cit. on p. 108).

[NIST15] National Institute of Standards and Technology (NIST). "SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions". In: Federal Information Processing Standards (FIPS) 202 (2015). DOI: 10.6028/nist.fips.202 (cit. on p. 17).

[NS09] Phong Q. Nguyen and Damien Stehlé. "Low-dimensional lattice basis reduction revisited". In: *ACM Trans. Algorithms* 5.4 (Nov. 2009). DOI: 10.1145/1597036.1597050 (cit. on p. 41).

[OW99] Paul C. van Oorschot and Michael J. Wiener. "Parallel Collision Search with Cryptanalytic Applications". In: *Journal of Cryptology* 12.1 (Jan. 1999), pp. 1–28. DOI: 10.1007/PL00003816 (cit. on p. 107).

[PR23] Aurel Page and Damien Robert. *Introducing Clapoti(s): Evaluating the isogeny class group action in polynomial time*. Cryptology ePrint Archive, Report 2023/1766. 2023. URL: https://eprint.iacr.org/2023/1766 (cit. on p. 82).

[PRR+25] Giacomo Pope, Krijn Reijnders, Damien Robert, Alessandro Sferlazza, and Benjamin Smith. "Simpler and Faster Pairings from the Montgomery Ladder". In: *CiC* 2.2 (2025), p. 29. DOI: 10.62056/ah2i893y6 (cit. on p. 61).

[PRSU26] Lorenz Panny, Ryan Rueger, Alessandro Sferlazza, and Aleksei Udovenko. "HyperSolver: Asymptotically and Concretely Accelerating the Delfs–Galbraith Attack using Isogeny Ladders". In: (2026). ASIACRYPT 2026 (to appear). URL: https://eprint.iacr.org/2026/1823 (cit. on p. 106).

[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. 5, 8, 106).

[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 pp. 3, 13).

[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. 61, 62).

[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. 56).

[Sch90] Claus-Peter Schnorr. "Efficient Identification and Signatures for Smart Cards". In: *CRYPTO'89*. Ed. by Gilles Brassard. Vol. 435. LNCS. Springer, New York, Aug. 1990, pp. 239–252. DOI: 10.1007/0-387-34805-0_22 (cit. on p. 12).

[Sch91] Claus-Peter Schnorr. "Efficient Signature Generation by Smart Cards". In: *Journal of Cryptology* 4.3 (Jan. 1991), pp. 161–174. DOI: 10.1007/BF00196725 (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. 26).

[SIK22] SIKE Team. *Supersingular Isogeny Key Encapsulation*. Specification, September 15, 2022. Sept. 2022. URL: https://sike.org/files/SIDH-spec.pdf (cit. on p. 107).

[Sil09] Joseph H. Silverman. *The arithmetic of elliptic curves*. Vol. 106. Springer, 2009. DOI: 10.1007/978-0-387-09494-6 (cit. on p. 8).

[Tan09] Seiichiro Tani. "Claw Finding Algorithms Using Quantum Walk". In: *Theoretical Computer Science* 410.50 (2009), pp. 5285–5297. DOI: 10.1016/j.tcs.2009.08.030 (cit. on p. 108).

[Vil92] Gilles Villard. "Parallel lattice basis reduction". In: *Papers from the international symposium on Symbolic and algebraic computation*. 1992, pp. 269–277. DOI: 10.1145/143242.143327 (cit. on p. 112).

[Voi21] John Voight. *Quaternion Algebras*. Springer Graduate Texts in Mathematics series, 2021, pp. XXIII, 885. DOI: 10.1007/978-3-030-56694-4 (cit. on p. 8).

[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, 8, 113).

[Wes24] Benjamin Wesolowski. *Random walks in number-theoretic cryptography*. École Normale Supérieure de Lyon, Aug. 2024. URL: https://bweso.com/hdr.pdf (cit. on p. 82).

[Wes26] Benjamin Wesolowski. *The supersingular isogeny problem in time and memory* $p^{1/3+o(1)}$. Cryptology ePrint Archive, Paper 2026/1486. 2026. URL: https://eprint.iacr.org/2026/1486 (cit. on pp. 4, 7, 106, 107).

[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. 57).

APPENDIX A

Proofs of Algorithm Correctness

In this chapter, we provide proofs of correctness for algorithms presented in Chapter 4 that have not yet been detailed in the literature.

A.1. HD Module

A.1.1. Generic $(2,2)$-isogenies

Assume that we want to compute a $(2,2)$-isogeny $\varphi\colon A \to B$ between non-product PPAS, and that we are given theta coordinates of 8-torsion points $K_1, K_2 \in A[8]$ such that $\ker(\varphi) = \langle [4]K_1, [4]K_2 \rangle$. We further assume that θ^A is a theta structure on A that is *adapted* to φ (see Definition 4.5.8).

Given such a theta structure, we can compute a compatible theta structure θ^B on B using the following proposition due to [DMPR24].

Proposition A.1.1. *Let A be a non-product PPAS with theta structure θ^A adapted to the $(2,2)$-isogeny $\varphi\colon A \to B$ with respect to a symplectic basis $\mathscr{B}$ of $A[8]$. Let θ^B be the theta structure on B induced by the symplectic basis $\varphi_*(\mathscr{B})$ of $B[4]$. Then, GenericCodomain returns $\widetilde{\theta}^B(0_B)^{\odot -1}$.*

Assuming *compatible* choices of theta structures θ^A and θ^B (as in Proposition A.1.1), we can use the following formula to evaluate a point $\theta^A(P)$:

$$\widetilde{\theta}^B(\varphi(P)) \cdot \widetilde{\theta}^B(0_B) = H\left(\theta^A(P)^{\odot 2}\right), \tag{21}$$

which implies

$$\widetilde{\theta}^B(\varphi(P)) \cdot \widetilde{\theta}^B(0_B) = H\left(\theta^A(P)^{\odot 2}\right) \odot \widetilde{\theta}^B(0_B)^{\odot -1}. \tag{22}$$

provided the coordinates of $\widetilde{\theta}^B(0_B)$ are invertible. GenericEval follows directly from Equation (22) and takes as input arrays of points $\theta^A(P)$ to evaluate along with $\widetilde{\theta}^B(0_B)^{\odot -1}$.

Recall that GenericCodomain outputs $\widetilde{\theta}^B(0_B)^{\odot -1}$. We can now give a short description of how it works. GenericCodomain takes as input $\theta^A(K_1), \theta^A(K_2)$. By [Dar25, Lemma 6.5.10], we have the following images under φ:

$$\widetilde{\theta}^B(\varphi(K_1)) = (x : x : y : y) \quad \text{and} \quad \widetilde{\theta}^B(\varphi(K_2)) = (z : t : z : t), \tag{23}$$

with $x, y, z, t \in \mathbb{F}_{p^2}$. From Equation (21), and letting $\widetilde{\theta}^B(0_B) = (\alpha : \beta : \gamma : \delta)$, we see that

$$\mathsf{Hadamard}(\mathsf{Squaring}(\theta^A(K_1))) = (\alpha x : \beta x : \gamma y : \delta y), \tag{24}$$

$$\mathsf{Hadamard}(\mathsf{Squaring}(\theta^A(K_2))) = (\alpha z : \beta t : \gamma z : \delta t). \tag{25}$$

From this, we can compute the projective point

$$\widetilde{\theta}^B(0_B)^{\odot -1} = (\alpha^{-1} : \beta^{-1} : \gamma^{-1} : \delta^{-1})$$

by Lemma A.1.2.(1). These compatible choices of theta structures θ^A and θ^B as in Proposition A.1.1 should be strictly ensured for the isogeny computation to be correct. To this end, VerifyKernel checks that $\widetilde{\theta}^B(\varphi(K_1))$ and $\widetilde{\theta}^B(\varphi(K_2))$ follow Equation (23), which can be checked with the conditions of Lemma A.1.2.(2). Putting this together gives us GenericCodomain.

Lemma A.1.2. *Let us denote $(x_i : y_i : z_i : t_i) \coloneqq H(\theta^A(K_i)^{\odot 2})$ for $i \in \{1,2\}$ and let $A^{-1} \coloneqq y_1 \cdot z_2 \cdot t_2$, $B^{-1} \coloneqq x_1 \cdot z_2 \cdot t_2$, $C^{-1} \coloneqq x_2 \cdot y_1 \cdot t_2$ and $D^{-1} \coloneqq x_1 \cdot y_2 \cdot z_2$. Then:*

(1) $(A^{-1} : B^{-1} : C^{-1} : D^{-1}) = (\alpha^{-1} : \beta^{-1} : \gamma^{-1} : \delta^{-1})$ in $\mathbb{P}^3(\mathbb{F}_{p^2})$.

(2) Equation (23) is satisfied by the images of $\theta^A(K_1)$ and $\theta^A(K_2)$ computed with GenericEval and $(A^{-1} : B^{-1} : C^{-1} : D^{-1})$ as input value for $\widetilde{\theta}^B(0_B)^{\odot -1}$ if only if $C^{-1} \cdot z_1 = D^{-1} \cdot t_1$.

Proof. **1.** We see that:

$$\begin{aligned} A^{-1} &= y_1 \cdot z_2 \cdot t_2 = \beta x \cdot \gamma z \cdot \delta t = \frac{\alpha\beta\gamma\delta xzt}{\alpha} \\ B^{-1} &= x_1 \cdot z_2 \cdot t_2 = \alpha x \cdot \gamma z \cdot \delta t = \frac{\alpha\beta\gamma\delta xzt}{\beta} \\ C^{-1} &= x_2 \cdot y_1 \cdot t_2 = \alpha z \cdot \beta x \cdot \delta t = \frac{\alpha\beta\gamma\delta xzt}{\gamma} \\ D^{-1} &= x_1 \cdot y_2 \cdot z_2 = \alpha x \cdot \beta t \cdot \gamma z = \frac{\alpha\beta\gamma\delta xzt}{\delta} \end{aligned} \tag{26}$$

so the projective equality $(A^{-1} : B^{-1} : C^{-1} : D^{-1}) = (\alpha^{-1} : \beta^{-1} : \gamma^{-1} : \delta^{-1})$ indeed holds.

2. The images of $\theta^A(K_1)$ and $\theta^A(K_2)$ computed with GenericEval and $(A^{-1} : B^{-1} : C^{-1} : D^{-1})$ as input value for $\widetilde{\theta}^B(0_B)^{\odot -1}$ are $(A^{-1} \cdot x_1 : B^{-1} \cdot y_1 : C^{-1} \cdot z_1 : D^{-1} \cdot t_1)$ and $(A^{-1} \cdot x_2 : B^{-1} \cdot y_2 : C^{-1} \cdot z_2 : D^{-1} \cdot t_2)$ respectively. They satisfy Equation (23) if and only if:

$$A^{-1} \cdot x_1 = B^{-1} \cdot y_1, \quad C^{-1} \cdot z_1 = D^{-1} \cdot t_1, \quad A^{-1} \cdot x_2 = C^{-1} \cdot z_2 \quad \text{and} \quad B^{-1} \cdot y_2 = D^{-1} \cdot t_2.$$

But by the definition of A^{-1}, B^{-1}, C^{-1} and D^{-1}, we have $A^{-1} \cdot x_1 = B^{-1} \cdot y_1 = x_1 y_1 z_2 t_2$, $A^{-1} \cdot x_2 = C^{-1} \cdot z_2 = x_2 y_1 z_2 t_2$ and $B^{-1} \cdot y_2 = D^{-1} \cdot t_2 = x_1 y_2 z_2 t_2$. The only non-trivial equality to check is the second one: $C^{-1} \cdot z_1 = D^{-1} \cdot t_1$. This completes the proof. □

A.1.2. The gluing $(2,2)$-isogeny

The goal of this section is to compute the first step of the isogeny chain computation: the gluing isogeny $\Phi_1 : E_1 \times E_2 \to A_1$. Recall that we need to find a theta structure θ^{A_0} that is adapted to Φ_1. Actually, by choosing a symplectic basis $\mathscr{B} := (S_1, S_2, T_1, T_2)$ of $A_0[2^{e+2}]$ adapted to Φ, we can not only ensure that θ^{A_0} is adapted to Φ_1 but also that $\theta^{A_{i-1}}$ is adapted to Φ_i and compatible with θ^{A_i} for all $1 \le i \le e$, as required to apply GenericCodomain at every generic step $i \ge 2$ (see Proposition A.1.1). By induction, this will be the case provided that θ^{A_i} is induced by the symplectic basis

$$\mathscr{B}_i := ([2^e]\Phi_i \circ \cdots \circ \Phi_1(S_1), [2^e]\Phi_i \circ \cdots \circ \Phi_1(S_2), [2^{e-i}]\Phi_i \circ \cdots \circ \Phi_1(T_1), [2^{e-i}]\Phi_i \circ \cdots \circ \Phi_1(T_2)), \tag{27}$$

of $A_i[4]$ for all $0 \le i \le e$, so that $\mathscr{B}_i = (\Phi_i)_*(\mathscr{B}_{i-1})$ for all $1 \le i \le e$.

At a high level, the gluing procedure works as follows. Let T_1 and T_2 be the input kernel points. We may assume that they are given in the following form:

$$T_1 := (P_1, [\nu_1]\varphi_2 \circ \varphi_1(P_1)) \quad \text{and} \quad T_2 := (Q_1, [\nu_1(1 - \nu_2 2^e)]\varphi_2 \circ \varphi_1(Q_1)), \tag{28}$$

with $\nu_1 \equiv N_1^{-1} \mod 2^{e+2}$, $\nu_2 \equiv N_2^{-1} \mod 2^{e+2}$ and (P_1, Q_1) a basis of $E_1[2^{e+2}]$. We then choose S_1 and S_2 accordingly:

$$S_1 := (0, [\nu_1(1 - \nu_2 2^e)]\varphi_2 \circ \varphi_1(Q_1)), \quad S_2 := (P_1, 0). \tag{29}$$

This defines a ζ-symplectic basis $\mathscr{B} := (S_1, S_2, T_1, T_2)$ of $A_0[2^{e+2}]$, where $\zeta := w_{2^{e+2}}(P_1, Q_1)$. From the $(X : Z)$-coordinates of E_1 and E_2, we can then compute the theta structure θ^{A_0} induced by the symplectic basis $\mathscr{B}_0 := [2^e]\mathscr{B}$ of $A_0[4]$ (adapted to Φ_1), using ThetaChangeOfCoordinates.

We then use GluingCodomain to determine the theta structure θ^{A_1} induced by the symplectic basis $\mathscr{B}_1 = (\Phi_1)_*(\mathscr{B}_0)$ of $A_1[4]$ given by Equation (27).

To proceed with the generic isogeny computations, we first need to evaluate Φ_1 given some data that represents θ^{A_1}. This data is not given by the dual theta null-point $\widetilde{\theta}^{A_1}(0_{A_1})$ because its last coordinate vanishes [Dar25, Lemma 6.5.14], which makes Equation (22) irrelevant to evaluate Φ_1 at any point. Instead, we use the images under Φ_1 of the 8-torsion points $[2^{e-1}]T_1, [2^{e-1}]T_2$ to represent θ^{A_1}, as an output of GluingCodomain. We can then evaluate Φ_1 with GluingEval. The evaluation algorithm is due to [Dup25].

A.1.2.1. More details on computing the codomain of a gluing $(2,2)$-isogeny. As we have seen, by Equation (23), there exists $x, y, z, t \in \mathbb{F}_{p^2}$ such that $\widetilde{\theta}^B(\varphi(K_1)) = (x : x : y : y)$ and $\widetilde{\theta}^B(\varphi(K_2)) = (z : t : z : t)$. We can determine these values projectively from Equation (24):

$$\mathsf{Hadamard}(\mathsf{Squaring}(\theta^A(K_1))) = (\alpha x : \beta x : \gamma y : 0),$$
$$\mathsf{Hadamard}(\mathsf{Squaring}(\theta^A(K_2))) = (\alpha z : \beta t : \gamma z : 0).$$

As in Lemma A.1.2, if we denote $(x_i : y_i : z_i : t_i) \coloneqq H(\theta^A(K_i)^{\odot 2})$ for $i \in \{1,2\}$, we then obtain in $\mathbb{P}^3(\mathbb{F}_{p^2})$:

$$\begin{aligned}
(\alpha : \beta : \gamma : 0) &= (x_1 \cdot x_2 : y_1 \cdot x_2 : x_1 \cdot z_2 : 0),\\
(\alpha^{-1} : \beta^{-1} : \gamma^{-1} : 0) &= (y_1 \cdot z_2 : x_1 \cdot z_2 : y_1 \cdot x_2 : 0),\\
\widetilde{\theta}^B(\varphi(K_1))^{\odot -1} &= (z_1 \cdot y_1 \cdot x_2 : z_1 \cdot y_1 \cdot x_2 : x_1 \cdot y_1 \cdot z_2 : x_1 \cdot y_1 \cdot z_2),\\
\widetilde{\theta}^B(\varphi(K_2))^{\odot -1} &= (y_2 \cdot x_1 \cdot z_2 : x_2 \cdot y_1 \cdot z_2 : y_2 \cdot x_1 \cdot z_2 : x_2 \cdot y_1 \cdot z_2).
\end{aligned}$$

A.1.2.2. Special points in gluing evaluation.

Lemma A.1.3. *If $P \in E_1 \times E_2$ is of the form $(P_1, 0)$ or $(0, P_2)$, then the last coordinate of $\widetilde{\theta}^B(\varphi(P))$ is 0.*

Proof. Equation (9) implies that:

$$\widetilde{\theta}^B(\varphi(P)) = H(\theta^A(P - W) \odot \theta^A(P + W)) \odot \widetilde{\theta}^B(W)^{\odot -1}.$$

Then, if $(u_i : v_i : w_i)$ are the barycentric coordinates of P_i and W_i for $i \in \{1,2\}$, we either have $P_1 = 0$, so $P_1 + W_1 = -(P_1 - W_1)$, so that $(u_1 - v_1 : w_1) = (u_1 + v_1 : w_1)$ *i.e.* $v_1 = 0$; or $P_2 = 0$, in which case $v_2 = 0$. In either case, we always have $v_1 v_2 = 0$. In addition, [Dup25, Appendix A] ensures that the last coordinate of $H(\theta^A(P - W) \odot \theta^A(P + W))$ is always a multiple of $v_1 v_2$. This completes the proof. □

A.1.3. Splitting the codomain

A.1.3.1. Theta structures on elliptic curves. There exist several conversion formulae between $(x : z)$ and theta coordinates, and they are all given by linear maps. In Section 4.5.7.1, we use the canonical conversion formula, in the sense that it outputs the theta structure θ^E induced by the basis of $E[4]$ formed by $P = (a + b : a - b)$ and $Q = (-1 : 1)$ in Montgomery $(x : z)$-coordinates. We may consider a non-canonical choice of theta structure ${\theta'}^E$ on E, induced by another basis (P', Q') of $E[4]$. Knowing the change of basis from (P', Q') to (P, Q), we can express the change of theta coordinates matrix $\mathbf{M} \in M_2(\mathbb{F}_{p^2})$ such that $\theta^E = \mathbf{M} \cdot {\theta'}^E$ with the formula from [Dar25, Theorem 6.2.10].

We can then express the canonical theta null-point $(a : b) = \theta^E(0)$ in terms of the non-canonical one $(a' : b') = {\theta'}^E(0)$, the Montgomery coefficient A_E in terms of $(a' : b')$ (using Section 4.5.7.1), and the change-of-coordinates matrix $\mathbf{N} \in M_2(\mathbb{F}_{p^2})$ from non-canonical theta coordinates ${\theta'_0}^E$, ${\theta'_1}^E$ to Montgomery $(x : z)$ coordinates $(x : z) = \mathbf{N} \cdot {\theta'}^E$, using Equation (10). Enumerating all possible changes of basis of $E[4]$ and the corresponding change-of-theta-coordinates matrices $\mathbf{M} \in M_2(\mathbb{F}_{p^2})$, we find the conversion formulae of Table 19.

In summary, given any theta structure ${\theta'}^E$ on E (represented by its theta null-point $(a' : b')$), we have 24 choices for the canonical theta structure θ^E and its associated conversion formulae. Taking into account the redundancies in Table 19, we may fix θ^E among the 6 canonical theta structures with theta null-point in

$$\{(a' : b'), (b' : a'), (a' + b' : a' - b'), (a' - b' : a' + b'), (a' + \mathbf{i}b' : a' - \mathbf{i}b'), (a' - \mathbf{i}b' : a' + \mathbf{i}b')\}$$

so that the corresponding Montgomery coefficient A_E is maximal for the order given by Definition 4.2.1.

A.1.3.2. Splitting change of theta coordinates. The next lemma tells us how to use the theta structure of $B = E_3 \times E_4$ to split it into a product theta structure. This is complicated by the fact that not all theta structures on B are product theta structures, so we need to first convert it into a product theta structure. Fortunately, if φ is the splitting $(2,2)$-isogeny of the chain $\Phi\colon E_1 \times E_2 \to E_3 \times E_4$, this is very simple.

Lemma A.1.4. *Assume that $\varphi\colon A \to B = E_3 \times E_4$ is the splitting $(2,2)$-isogeny of the chain $\Phi\colon E_1 \times E_2 \to E_3 \times E_4$ computed with Isogeny22Chain with kernel inputs given by Equation (28). Let us denote $\theta^B \coloneqq (x : y : z : t)$ and $\widetilde{\theta}^B \coloneqq (\widetilde{x} : \widetilde{y} : \widetilde{z} : \widetilde{t})$. Then there exists a product theta structure $\theta^{E_3} \times \theta^{E_4} \coloneqq (x' : y' : z' : t')$ on $B = E_3 \times E_4$ such that:*

$$(x' : y' : z' : t') = (x + y : x - y : z - t : z + t) = (\widetilde{x} + \widetilde{z} : \widetilde{y} + \widetilde{t} : \widetilde{y} - \widetilde{t} : \widetilde{x} - \widetilde{z}).$$

PROOF. The theta structure θ^B that we obtain at the end of the $(2,2)$-isogeny chain computation is induced by the symplectic basis $\Phi_*(\mathscr{B}) \coloneqq ([2^e]\Phi(S_1), [2^e]\Phi(S_2), \Phi(T_1), \Phi(T_2))$ of $B[4]$, where $\mathscr{B} \coloneqq (S_1, S_2, T_1, T_2)$ is the symplectic basis of $(E_1 \times E_2)[2^{e+2}]$ given by Equation (29):

$$S_1 \coloneqq (0, [\nu_1(1-\nu_2 2^e)]\varphi_2 \circ \varphi_1(Q_1)), \quad S_2 \coloneqq (P_1, 0)$$
$$T_1 \coloneqq (P_1, [\nu_1]\varphi_2 \circ \varphi_1(P_1)), \quad T_2 \coloneqq (Q_1, [\nu_1(1-\nu_2 2^e)]\varphi_2 \circ \varphi_1(Q_1)),$$

where (P_1, Q_1) is a basis of $E_1[2^{e+2}]$, $\nu_1 \equiv N_1^{-1} \mod 2^{e+2}$ and $\nu_2 \equiv N_2^{-1} \mod 2^{e+2}$. We do not expect this theta structure θ^B to be a product in general. However, from $\Phi_*(\mathscr{B})$, we can compute the change of theta coordinates matrix from θ^B to a product theta structure $\theta^{E_3} \times \theta^{E_4}$ as follows.

By Equation (29), we have:

$$\begin{aligned}
\Phi(S_1) &= \Phi(0, [\nu_1(1-\nu_2 2^e)]\varphi_2 \circ \varphi_1(Q_1)) = [\nu_1(1-\nu_2 2^e)](\widehat{\varphi}_2 \circ \varphi_2 \circ \varphi_1(Q_1), \widehat{\psi}_1 \circ \varphi_2 \circ \varphi_1(Q_1)) \\
&= [\nu_1(1-\nu_2 2^e)]([N_2]\varphi_1(Q_1), [N_1]\psi_2(Q_1)) = ([\nu_1(N_2 - 2^e)]\varphi_1(Q_1), [1-\nu_2 2^e]\psi_2(Q_1)) \\
&= ([-N_1\nu_1]\varphi_1(Q_1), [\nu_2(N_2-2^e)]\psi_2(Q_1)) = (-\varphi_1(Q_1), -[\nu_2 N_1]\psi_2(Q_1)) \\
\Phi(S_2) &= \Phi(P_1, 0) = (\varphi_1(P_1), -\psi_2(P_1)) \\
\Phi(T_1) &= \Phi(P_1, [\nu_1]\varphi_2 \circ \varphi_1(P_1)) = (\varphi_1(P_1) + [\nu_1]\widehat{\varphi}_2 \circ \varphi_2 \circ \varphi_1(P_1), -\psi_2(P_1) + [\nu_1]\widehat{\psi}_1 \circ \varphi_2 \circ \varphi_1(P_1)) \\
&= ([1+\nu_1 N_2]\varphi_1(P_1), -\psi_2(P_1) + [\nu_1 N_1]\psi_2(P_1)) = ([\nu_1 2^e]\varphi_1(P_1), 0) \\
\Phi(T_2) &= \Phi(Q_1, [\nu_1(1-\nu_2 2^e)]\varphi_2 \circ \varphi_1(Q_1)) \\
&= (\varphi_1(Q_1) + [\nu_1(1-\nu_2 2^e)]\widehat{\varphi}_2 \circ \varphi_2 \circ \varphi_1(Q_1), -\psi_2(Q_1) + [\nu_1(1-\nu_2 2^e)]\widehat{\psi}_1 \circ \varphi_2 \circ \varphi_1(P_1)) \\
&= ([1+N_2\nu_1(1-\nu_2 2^e)]\varphi_1(Q_1), [-1+N_1\nu_1(1-\nu_2 2^e)]\psi_2(Q_1)) = (0, -[\nu_2 2^e]\psi_2(Q_1))
\end{aligned}$$

Now, let θ^{E_3} and θ^{E_4} be the theta structures induced by the basis $\mathscr{B}_3 \coloneqq ([2^e]\varphi_1(P_1), [2^e\nu_1]\varphi_1(Q_1))$ and $\mathscr{B}_4 \coloneqq ([2^e]\psi_2(P_1), [2^e\nu_2]\psi_2(Q_1))$ of $E_3[4]$ and $E_4[4]$ respectively. We then have:

$$([2^e]\varphi_1(P_1), 0) = [N_1]\Phi(T_1), \quad (0, [2^e]\psi_2(P_1)) = -[2^e]\Phi(S_2) + [N_1]\Phi(T_1),$$
$$([2^e\nu_1]\varphi_1(Q_1), 0) = -[2^e\nu_1]\Phi(S_1) + \Phi(T_2), \quad (0, [2^e\nu_2]\psi_2(Q_1)) = -\Phi(T_2).$$

We remark that with $\mathscr{B}_3$, $\mathscr{B}_4$ as above, $\theta^{E_3} \times \theta^{E_4}$ is induced by the ζ_4-symplectic basis of $B[4]$:

$$\mathscr{B}_3 \times \mathscr{B}_4 \coloneqq ((U_3, 0), (0, U_4), (V_3, 0), (0, V_4)).$$

Therefore, the change of basis matrix from $\Phi_*(\mathscr{B})$ to $\mathscr{B}_3 \times \mathscr{B}_4$ is of the form:

$$\begin{pmatrix} 0 & 0 & -N_1 & 0 \\ 0 & -1 & 0 & 0 \\ N_1 & N_1 & 0 & 0 \\ 0 & 0 & 1 & -1 \end{pmatrix},$$

where we used the fact that $N_1 \equiv \nu_1 \mod 4$. Enumerating all possible values for $N_1 \mod 4$ (1 and 3) and using [Dar25, Theorem 6.2.10], we obtain the following change of theta coordinate matrix:

$$\mathbf{M} = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 1 & -1 & 0 & 0 \\ 0 & 0 & 1 & -1 \\ 0 & 0 & 1 & 1 \end{pmatrix}$$

such that $\theta^{E_3} \times \theta^{E_4} = \mathbf{M} \cdot \theta^B$, so that $\theta^{E_3} \times \theta^{E_4} = \mathbf{M} \cdot H(\widetilde{\theta}^B)$. The result follows. □

TABLE 19. Enumeration of elliptic curve theta structures and associated conversion formulae to Montgomery $(x : z)$-coordinates.

M	$(a : b)$	A_E	**N**
$\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$	$(a' : b')$	$-2\dfrac{a'^4 + b'^4}{a'^4 - b'^4}$	$\begin{pmatrix} b' & a' \\ -b' & a' \end{pmatrix}$
$\begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$	$(a' : -b')$		
$\begin{pmatrix} 1 & 0 \\ 0 & \mathbf{i} \end{pmatrix}$	$(a' : \mathbf{i}b')$		
$\begin{pmatrix} 1 & 0 \\ 0 & -\mathbf{i} \end{pmatrix}$	$(a' : -\mathbf{i}b')$		
$\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$	$(b' : a')$	$2\dfrac{a'^4 + b'^4}{a'^4 - b'^4}$	$\begin{pmatrix} b' & a' \\ b' & -a' \end{pmatrix}$
$\begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}$	$(b' : -a')$		
$\begin{pmatrix} 0 & 1 \\ \mathbf{i} & 0 \end{pmatrix}$	$(b' : \mathbf{i}a')$		
$\begin{pmatrix} 0 & 1 \\ -\mathbf{i} & 0 \end{pmatrix}$	$(b' : -\mathbf{i}a')$		
$\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$	$(a' + b' : a' - b')$	$-\dfrac{a'^4 + 6a'b' + b'^4}{2(a'^3b' + a'b'^3)}$	$\begin{pmatrix} a' & -b' \\ b' & -a' \end{pmatrix}$
$\begin{pmatrix} 1 & 1 \\ -1 & 1 \end{pmatrix}$	$(a' + b' : b' - a')$		
$\begin{pmatrix} 1 & 1 \\ \mathbf{i} & -\mathbf{i} \end{pmatrix}$	$(a' + b' : \mathbf{i}(a' - b'))$		
$\begin{pmatrix} 1 & 1 \\ -\mathbf{i} & \mathbf{i} \end{pmatrix}$	$(a' + b' : \mathbf{i}(-a' + b'))$		
$\begin{pmatrix} 1 & -1 \\ 1 & 1 \end{pmatrix}$	$(a' - b' : a' + b')$	$\dfrac{a'^4 + 6a'b' + b'^4}{2(a'^3b' + a'b'^3)}$	$\begin{pmatrix} a' & -b' \\ -b' & a' \end{pmatrix}$
$\begin{pmatrix} 1 & -1 \\ -1 & -1 \end{pmatrix}$	$(a' - b' : -(a' + b'))$		
$\begin{pmatrix} 1 & -1 \\ \mathbf{i} & \mathbf{i} \end{pmatrix}$	$(a' - b' : \mathbf{i}(a' + b'))$		
$\begin{pmatrix} 1 & -1 \\ -\mathbf{i} & -\mathbf{i} \end{pmatrix}$	$(a' - b' : -\mathbf{i}(a' + b'))$		
$\begin{pmatrix} 1 & \mathbf{i} \\ 1 & -\mathbf{i} \end{pmatrix}$	$(a' + \mathbf{i}b' : a' - \mathbf{i}b')$	$\dfrac{\mathbf{i}(a'^4 - 6a'b' + b'^4)}{2(a'^3b' - a'b'^3)}$	$\begin{pmatrix} a' & b' \\ \mathbf{i}b' & -\mathbf{i}a' \end{pmatrix}$
$\begin{pmatrix} 1 & \mathbf{i} \\ \mathbf{i} & 1 \end{pmatrix}$	$(a' + \mathbf{i}b' : \mathbf{i}a' + b')$		
$\begin{pmatrix} 1 & \mathbf{i} \\ -\mathbf{i} & -1 \end{pmatrix}$	$(a' + \mathbf{i}b' : -\mathbf{i}a' - b')$		
$\begin{pmatrix} 1 & \mathbf{i} \\ -1 & \mathbf{i} \end{pmatrix}$	$(a' + \mathbf{i}b' : -a' + \mathbf{i}b')$		
$\begin{pmatrix} 1 & -\mathbf{i} \\ 1 & \mathbf{i} \end{pmatrix}$	$(a' - \mathbf{i}b' : a' + \mathbf{i}b')$	$-\dfrac{\mathbf{i}(a'^4 - 6a'b' + b'^4)}{2(a'^3b' - a'b'^3)}$	$\begin{pmatrix} a' & b' \\ -\mathbf{i}b' & \mathbf{i}a' \end{pmatrix}$
$\begin{pmatrix} 1 & -\mathbf{i} \\ -\mathbf{i} & 1 \end{pmatrix}$	$(a' - \mathbf{i}b' : -\mathbf{i}a' + b')$		
$\begin{pmatrix} 1 & -\mathbf{i} \\ \mathbf{i} & -1 \end{pmatrix}$	$(a' - \mathbf{i}b' : \mathbf{i}a' - b')$		
$\begin{pmatrix} 1 & -\mathbf{i} \\ -1 & -\mathbf{i} \end{pmatrix}$	$(a' - \mathbf{i}b' : -(a' + \mathbf{i}b'))$		

APPENDIX B

Generating Global Constants

We provide the algorithms used to generate global constants for which clear formulas were not provided. These include the basis $\mathcal{B}_0 = (P_0, Q_0)$ of $E_0[2^f]$, RICofactor, EIBox. See Section 4.7 for more details.

Algorithm B.1 GenerateBasis$\mathcal{B}_0(p)$

Input: A prime p with $p \equiv 3 \bmod 4$
Output: The basis $\mathcal{B}_0 = (P_0, Q_0)$ of $E_0[2^f]$
Convention: $\mathbb{F}_{p^2}$ is totally ordered by $a + b\mathsf{i} \leq c + d\mathsf{i} :\Leftrightarrow (a, b) \leq_{\text{lex}} (c, d)$, where elements of $\mathbb{F}_p$ are compared through their minimal non-negative representatives

$\mathbb{F}_{p^2} \leftarrow \mathbb{F}_p(\mathsf{i})$ with $\mathsf{i}^2 = -1$
$E_0 : y^2 = x^3 + x$ over $\mathbb{F}_{p^2}$
$f \leftarrow \mathsf{DyadicValuation}(p + 1)$
$m \leftarrow 0$
$P_0 \leftarrow \perp$
while `true` **do**
  $m \leftarrow m + 1, \quad x \leftarrow 1 + \mathsf{i}\, m$
  if $x^3 + x$ is not a square in $\mathbb{F}_{p^2}$ **then**
    continue
  $y \leftarrow \sqrt{x^3 + x}$
  $R \leftarrow (x, y) \in E_0(\mathbb{F}_{p^2})$
  if $-y \leq_{\text{lex}} y$ **then**
    $R \leftarrow -R$
  $n \leftarrow \text{ord}(R)$
  if $2^f \nmid n$ **then**
    continue
  $R \leftarrow [n/2^f]R$
  if $P_0 == \perp$ **then**
    $P_0 \leftarrow R$
  else if $e_{2^f}(P_0, R)^{2^{f-1}} == -1$ **then** // $e_{2^f}(\cdot, \cdot)$ is the Weil pairing
    $Q_0 \leftarrow R$
    break
if $[2^{f-1}]Q_0 == (0, 0)$ **then**
  pass
else if $[2^{f-1}]P_0 == (0, 0)$ **then**
  $(P_0, Q_0) \leftarrow (Q_0, P_0)$
else
  $Q_0 \leftarrow Q_0 + P_0$
return (P_0, Q_0)

Note. In Algorithm B.2 and Algorithm B.3, we use $\log_2(p)$ and $\log_2(M)$ instead of $\log(p)$ and $\log(M)$ as discussed in Section 9.3, but this is not problematic as the computed values satisfy the required inequalities.

Algorithm B.2 ElBox(p, λ)

Input: The base prime p
The security parameter λ

Output: The least integer b such that $\left(1 - \frac{1}{2\log(p)}\right)^{(b+1)^4} < 2^{-\lambda}$ // See Section 9.3

1: $b \leftarrow \lceil \log_2(p) \rceil$
2: $b \leftarrow \log_2\left(1 - \frac{1}{2b}\right)$
3: $b \leftarrow -\lambda / b$
4: $b \leftarrow b^{1/4} - 1$
5: $b \leftarrow \lceil b \rceil$
6: **return** b

Algorithm B.3 RICofactor$(p, e_{\mathsf{rsp}}, \lambda)$

Input: The base prime p
The response exponent e_{rsp}
The security parameter λ

Output: A prime q // See Section 9.3

1: **function** L(n, b)
2: $\quad \mathsf{val} \leftarrow \log_2\left(1 - \frac{1}{2b}\right)$
3: $\quad \mathsf{val} \leftarrow n \cdot \mathsf{val}$
4: $\quad \mathsf{val} \leftarrow \lceil \mathsf{val} \rfloor$
5: $\quad$ **return** val
6: $m \leftarrow \lceil \log_2(p) \rceil - e_{\mathsf{rsp}} + 2$
7: **while** L$(\lfloor 2^{e_{\mathsf{rsp}}+m}/p \rfloor, e_{\mathsf{rsp}} + m) > -\lambda$ **do**
8: $\quad m \leftarrow m + 1$
9: $q \leftarrow$ `next_prime`(2^m)
10: **return** q

APPENDIX P

Other parameter sets of interest

As explained in Section 8.2.1.2, for our choice of parameters, we made very conservative choices in the evaluation of the amortized cost of each j-invariant computed in the meet-in-the-middle attack. We expect the practical cost (e.g., of Cantor–Zassenhaus as in Remark 8.2.1) to be substantially higher than these conservative estimates.

In this section we list more aggressive choices of parameters that may still fulfill the NIST security requirements. We also list even more conservative choices that, for the meet-in-the-middle attack, require a larger number of stored j-invariants than the bounds of $w = 2^{80}, 2^{120}, 2^{160}$ used by Section 8.2.1.2, for NIST levels I, III, and V respectively.

We encourage the community to implement these other primes and perform more cryptanalysis on them. The bold primes refer to the primes selected for our NIST-I, NIST-III and NIST-V parameter sets, respectively.

- For NIST security level I: $p_{306.3} = 3 \cdot 2^{306} - 1$, $p_{305.55} = 55 \cdot 2^{305} - 1$, $p_{313.69} = 69 \cdot 2^{313} - 1$, $\mathbf{p_{324.3} = 3 \cdot 2^{324} - 1}$, $p_{343.9} = 9 \cdot 2^{343} - 1$, $p_{376.65} = 65 \cdot 2^{376} - 1$;
- For NIST security level III: $p_{448.3} = 3 \cdot 2^{448} - 1$, $p_{470.3} = 3 \cdot 2^{470} - 1$, $\mathbf{p_{500.27} = 27 \cdot 2^{500} - 1}$, $p_{521.1} = 2^{521} - 1$, $p_{574.27} = 27 \cdot 2^{574} - 1$;
- For NIST security level V: $p_{607.1} = 2^{607} - 1$, $p_{612.17} = 17 \cdot 2^{612} - 1$, $\mathbf{p_{664.17} = 17 \cdot 2^{664} - 1}$, $p_{665.19} = 19 \cdot 2^{665} - 1$, $p_{767.43} = 43 \cdot 2^{767} - 1$.

Table 20 gives the main properties of each prime.

TABLE 20. List of alternate prime suggestions. Primes in bold refer to our selected primes. Sizes are in bytes.

p	$\log p$	λ	Public key	Secret key	Signature
$p_{306.3}$	308	128	79	260	192
$p_{305.55}$	311	128	79	260	192
$p_{313.69}$	320	128	81	265	198
$\mathbf{p_{324.3}}$	326	128	83	270	200
$p_{343.9}$	347	128	89	285	214
$p_{376.65}$	383	128	97	305	230
$p_{448.3}$	460	192	117	387	286
$p_{470.3}$	472	192	119	392	288
$\mathbf{p_{500.27}}$	505	192	129	417	306
$p_{521.1}$	521	192	133	427	314
$p_{574.27}$	579	192	147	462	344
$p_{607.1}$	607	256	153	509	373
$p_{612.17}$	617	256	157	519	378
$\mathbf{p_{664.17}}$	669	256	169	549	406
$p_{665.19}$	670	256	169	549	406
$p_{767.43}$	773	256	195	614	456