QIMEN-PRISM

Algorithm specifications & supporting documentation

(Version 1.0)

- Yu Yu, Shanghai Jiao Tong University, yyuu@sjtu.edu.cn
- Wouter Castryck, COSIC, KU Leuven, wouter.castryck@esat.kuleuven.be
- Mingjie Chen, COSIC, KU Leuven, mjchennn555@gmail.com
- Arthur Herlédan Le Merdy, COSIC, KU Leuven, arthur.herledanlemerdy@esat.kuleuven.be
- Zhi Hu, Central South University, huzhi_math@csu.edu.cn
- Zhuo Huang, Shanghai Jiao Tong University, sh1kaku@sjtu.edu.cn
- Riccardo Invernizzi, COSIC, KU Leuven, riccardo.invernizzi@esat.kuleuven.be
- Yi-Fu Lai, Shanghai Jiao Tong University, yifu.lai@sjtu.edu.cn
- Dania Lazzarini, Université Libre de Bruxelles, dania.lazzarini@ulb.be
- Kaizhan Lin, Fudan University, linkzh@fudan.edu.cn
- Xuzhe Liu, Central South University, luuuxuuz@gmail.com
- Luciano Maino, University of Birmingham, mainoluciano.96@gmail.com
- Krijn Reijnders, COSIC, KU Leuven, reijnders.krijn@gmail.com
- Frederik Vercauteren, COSIC, KU Leuven, frederik.vercauteren@esat.kuleuven.be
- Yunqi Wen, Central South University, 232103006@csu.edu.cn

July 10, 2026

Contents

CHAPTER 1

Introduction

The QIMEN (**Q**uaternion and **I**sogeny **M**achinery over **EN**domorphism ring) cryptographic suite represents a comprehensive, next-generation framework designed to secure digital infrastructures against both classical and quantum adversaries. The suite is composed of two primary building blocks: a key-encapsulation mechanism, designated as QIMEN-PIKE, and a digital signature scheme, designated as QIMEN-PRISM. Both schemes are constructed upon the foundational hardness of the supersingular isogeny path and endomorphism ring problems, which have emerged as resilient mathematical paradigms within post-quantum cryptography. There are many sections that are aligned between the specifications of QIMEN-PIKE and QIMEN-PRISM, as they are common building blocks to both.

This document describes the digital signature scheme QIMEN-PRISM.

1.1 Design rationale

The core design of QIMEN-PRISM builds upon the digital signature scheme salt-PRISM [BBC+26] (best paper at PKC'25). It follows the same mathematical hardness assumption and protocol architecture.

Mathematical problem. The security of QIMEN-PRISM relies on a classic computational problem in elliptic-curve arithmetic: given a supersingular elliptic curve E and a prescribed large prime q, compute an isogeny of degree q from E without knowing its endomorphism ring. Explicit isogeny computation has been studied since Vélu's formulas in 1971 [Vél71], followed by extensive work on torsion methods, modular polynomials, isogeny graphs, and endomorphism rings [Cou96, Elk98, Koh96, LM00, BCL08, DG16]. Isogenies whose degrees factor into small primes can be computed as chains of low-degree maps, but this approach does not apply to the large prime degree q. After more than five decades of algorithmic development, the best general algorithms require $\widetilde{O}(\sqrt{q})$ field operations [BDLS20], which is exponential in the bit length of q.

Core design idea. To utilize this problem in a cryptographic construction, QIMEN-

PRISM exploits the gap between computing a large-prime-degree isogeny with and without knowledge of the endomorphism ring. When the endomorphism ring of a supersingular elliptic curve E is known, such an isogeny can be constructed efficiently. Accordingly, the public key is only the curve E, while the secret key is its endomorphism ring.

The underlying identification protocol is simple. The verifier samples a random large prime q as the challenge, and the prover uses the secret endomorphism ring to construct an isogeny of degree q from E. To allow a verifier who does not know the endomorphism ring to check this isogeny efficiently, the prover embeds it, through an algorithmic realization of Kani's Lemma, into a two-dimensional isogeny of smooth degree $(2^a, 2^a)$ between products of elliptic curves, and returns the kernel of this two-dimensional isogeny as the response. Using only the public curve and the challenge q, the verifier reconstructs the resulting smooth-degree isogeny chain with theta-coordinate arithmetic and uses pairings to verify its consistency with the underlying prime-degree isogeny.

Hash-and-sign paradigm. Following the hash-and-sign paradigm, by constructing a hash-to-prime function, the identification protocol is transformed into a simple signature scheme. By hashing a message to a prime locally, the signer can generate the proof as the signature of the message without interaction.

Parameter selection and failure analysis. The three parameter sets of QIMEN-PRISM, denoted by NGCC-1, NGCC-2, and NGCC-3, target classical security strengths of 128, 256, and 512 bits, respectively, and quantum security strengths of 80, 128, and 256 bits. For all three parameter sets, the overall failure probability is bounded above by 2^{-128}. In Section 8.4, we justify the parameter choices for each security level in accordance with the NGCC submission requirements. A detailed analysis of the failure probabilities is provided in Chapter 9.

1.1.1 Improvements over salt-PRISM

While building on the core design of salt-PRISM [BBC+26], QIMEN-PRISM introduces several theoretical and engineering innovations that distinguish it from prior academic prototypes. These differences are summarized as follows.

1. **Faster core subroutines.** The additional mechanisms introduced in the core subroutines, together with the refined parameter choices of QIMEN-PRISM, allow suitable intermediate candidates to be recycled across successive stages, including Cornacchia computations, and norm-equation solving (see for instance Remarks 3.2.1, 3.2.2 and 3.3.1 for more details). This avoids restarting independent searches, reduces rejection loops and redundant primality tests in salt-PRISM, and thereby substantially improves the overall performance without compromising security. In summary, the improvements reduce key generation and verification by approximately 6–9.5%, and signing by approximately 27–40% depending on the security level.

2. **Fast signature compression.** QIMEN-PRISM reuses geometric values and change-of-basis matrices already computed during isogeny evaluation. These intermediate values allow part of the signature data to be reconstructed rather than transmitted explicitly, reducing the signature size by a factor of approximately 1.5 to 1.7 with only mild overhead, depending on the security level. The compression introduces only approximately 2% additional signing cost and 15% additional verification cost.

3. **Provably secure primality testing.** The hash-to-prime procedure uses a fixed number of Miller–Rabin iterations for each security level. The number of iterations is chosen so that the probability of accepting a composite integer is at most $2^{-\lambda_c}$, preventing the hash-to-pseudoprime forgery attacks described in [CGMT26].

4. **Assembly-level field arithmetic kernels.** The optimized implementation provides x86-64 assembly routines using BMI2 and ADX instructions. These routines reduce the cost of modular arithmetic, resulting in runtime reductions of approximately $14, 30\%$ for key generation, $15, 20\%$ for signing, and 40% for verification compared with the portable C implementation for NGCC-1 and NGCC-2 security level respectively.

1.2 Feature Statements

1.2.1 Innovativeness

Most existing isogeny-based digital signature schemes follow a sigma-protocol-and-Fiat–Shamir paradigm, as exemplified by [DKL+20, DLLW23, DLRW24, BDD+24, NOC+24] using the SQIsign diagram originated in [DKL+20]. Among these constructions, the work of [BDD+24] forms a principal basis of SQIsign [AAA+25], which is currently the only isogeny-based digital signature scheme undergoing international standardization evaluation and has advanced to the third round of the NIST Additional Digital Signatures process.

In contrast, salt-PRISM, and consequently QIMEN-PRISM, follow a distinct hash-and-sign architecture. Moreover, their principal innovations lie in the specific signing relation and public verification mechanism introduced within this paradigm. In the following, we refer to QIMEN-PRISM throughout, while noting that these protocol-level innovations originate from salt-PRISM.

- **Fresh prime-degree signing relation.** QIMEN-PRISM is the first isogeny-based signature scheme in which every message and salt determine a fresh large prime q, used as the degree of the one-dimensional isogeny underlying the signature. Each signature is therefore tied to a newly generated prime-degree isogeny instance.
- **Kani-and-pairing verification for randomized-degree signatures.** QIMEN-PRISM is the first isogeny-based signature scheme to combine an algorithmic realization of Kani's Lemma with pairing checks to verify signatures whose underlying

large prime degree is freshly randomized for each signature. Kani's Lemma embeds the prime-degree isogeny into a smooth $(2^a, 2^a)$-isogeny between products of elliptic curves, while the pairing check certifies that the reconstructed two-dimensional isogeny contains the prescribed prime-degree component.

Together, these features establish QIMEN-PRISM as an independent isogeny-based signature architecture rather than a modification or parameterization of the SQIsign framework undergoing international standardization evaluation.

1.2.2 Simplicity

A primary architectural advantage of QIMEN-PRISM is its fundamental design simplicity, particularly when contrasted with the state-of-the-art isogeny signature scheme SQIsign [AAA+25]. SQIsign is derived from a three-round interactive identification protocol consisting of a commitment phase, a challenge phase, and a response phase, which is converted to a signature via the Fiat-Shamir transform. This results in a much more complex signing operation, directly affecting the security analysis and the flexibility of the design. In contrast, QIMEN-PRISM bypasses the commitment phase entirely. It operates as a direct trapdoor one-way function in a hash-and-sign configuration.

The security proofs of SQIsign and its high-dimensional variants are heavily reliant on powerful, ad-hoc simulated oracles that are required to argue the Honest-Verifier Zero-Knowledge property by simulating valid transcripts without the secret key. In contrast, QIMEN-PRISM reduces directly to the problem of computing large prime-degree isogenies in the standard model (or standard Quantum Random Oracle Model), utilizing clear reductionist arguments that do not require complex oracle overlays.

1.2.3 Flexibility and Compatibility

The minimalist and modular design of QIMEN-PRISM provides flexibility across diverse engineering environments and compatibility with modern cryptographic enhancement transforms:

- **Extensible hash/XOF backends.** The protocol separates the high-level hash-to-prime routine from the underlying extendable output function (XOF). For standard global deployments, it natively supports a SHAKE256 instantiation. For compliance with national commercial cryptography standards, it seamlessly integrates a pseudoXOF construction driven by the SM3 cryptographic hash algorithm (conforming to GB/T 32905-2016). This allows the same protocol machinery to be compiled for different regulatory domains via simple build-time switches.
- **BUFF transform compatibility.** QIMEN-PRISM is fully compatible with the BUFF transform proposed in [CDF+21, DFH+24, FHK25] to achieve the security notions at the cost of $2\lambda_c$-bit overhead in the signature and one hash computation.

The core scheme can be fortified to achieve advanced security properties including *exclusive-ownership*, *message-bound signatures* and *non-resignability*.

- **Flexible point compression modes.** The representation of the signature isogeny can be tailored based on the bandwidth and computational profile of the target platform. It supports a coordinate-centric mode (ideal for minimized verification overhead, requiring a single field inversion) as well as a basis-coefficient mode that compresses the signature size down to $6\lambda_c + 4a$ bits at the cost of providing basis generation hints.

1.2.4 Extensibility

Perhaps the most significant advantage of the QIMEN-PRISM approach over conventional isogeny-based signatures is its exceptional extensibility. The intricate structure of SQIsign has historically created a bottleneck for the development of advanced multi-party or structured cryptographic protocols, with only a small number of complex primitives proposed after years of scrutiny. In contrast, the direct hash-and-sign nature of QIMEN-PRISM, naturally generalizes to several advanced applications:

- **Signatures with randomizable keys.** As shown in [BBS+26], QIMEN-PRISM can be extended to allow key randomization and message adaptation. In particular, this second property seems hard to obtain for SQIsign. This extension achieves unlinkability against unbounded adversaries and remains unforgeable under the same assumptions as QIMEN-PRISM.
- **Well-formedness proofs.** QIMEN-PRISM supports efficient proofs of well-formedness for both public keys and signatures [LM26]. These proofs allow a prover to demonstrate that a public key or a signature is valid without revealing the underlying information. This property facilitates the secure use of QIMEN-PRISM as a building block in more advanced cryptographic protocols, where malformed public keys or signatures could otherwise invalidate the security analysis.
- **Threshold Signatures.** Even more notably, QIMEN-PRISM is suitable to build a T-out-of-N threshold signature protocol [Thr], a property which SQIsign failed to achieve so far. Furthermore, [Thr] is the first isogeny-based threshold signature not relying on group actions, and hence not susceptible to subexponential quantum attacks. For this reason, its combined public key and signature size and its communication cost are remarkably small compared to its competitors.

1.2.5 Performance

- **Compactness.** QIMEN-PRISM has compact signature size. At the same security level, compared with the Falcon and the standardized ML-DSA [PFH+22, LDK+22], QIMEN-PRISM achieves public keys and signatures smaller by a factor of 13.9, resp.

3.8, compared to Falcon, and a factor of 20.1, resp. 13.9, compared to ML-DSA (see Table 1.1 below). This makes QIMEN-PRISM well suited for bandwidth-constrained applications like certificate chains.

Table 1.1: Comparison of public-key and signature sizes, in bytes, for NIST-standardized or to-be-standardized post-quantum signature schemes at the security level NIST-5, which is the same as QIMEN-PRISM of NGCC-2.

Scheme	Public key	Signature
QIMEN-PRISM (NGCC-2)	129 **B**	334 **B**
FN-DSA-1024 (Falcon)	1793 B (13.9×)	1280 B (3.8×)
ML-DSA-87 (Dilithium)	2592 B (20.1×)	4627 B (13.9×)
SLH-DSA-SHAKE-256s (SPHINCS$^+$)	64 B (0.5×)	29 792 B (89.2×)

- **Reasonable signing time.** While slower than ML-DSA and Falcon, QIMEN-PRISM achieves reasonable efficiency that makes it practical for most real-world applications. For example, our 64-bit implementation at the NGCC-1 security level requires approximately 24.5 ms for signing and 3.6 ms for verification on an Intel Ultra 9 CPU running at 2.7 GHz (see Table 7.4).
- **Faster verification.** QIMEN-PRISM is attractive due to its much faster verification than signing.
  This makes QIMEN-PRISM especially suitable for applications where signatures are generated offline but verified at large scale, such as certificate chains, code updates, and transparency-related signatures.

Table 1.2: Comparison of reference isogeny-based signature performance at NIST security category 5 in 10^6 CPU cycles at the same security level as QIMEN-PRISM on Intel Ultra 9 CPU (2.7GHz).

Scheme	KeyGen	Sign	Verify
QIMEN-PRISM (NGCC-2)	130.24	142.46	41.72
SQIsign+[BCRSE$^+$26] (NIST-5)	129.94 (1.0x)	332.74 (2.3x)	35.87 (0.9x)

1.2.6 Diversity of assumptions

QIMEN-PRISM relies on isogeny-based assumptions, which are fundamentally different from those underlying lattice-based and code-based constructions. Therefore, QIMEN-PRISM contributes to the diversity of the quantum-safe cryptographic portfolio for the quantum-safe infrastructure. The endomorphism ring problem enjoys a simple worst-case

to average-case self-reduction for the uniform distribution. QIMEN-PRISM 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.

Star superscript

Isogeny-based cryptography is a highly involved field, with many protocols that build on previous material. As such, much of the lower-level arithmetic for finite fields, elliptic curves, and quaternions is standard and shared between implementations and documentation, to achieve consistency with the prior work upon which such schemes build. To ensure consistency, we have chosen to reuse parts of the detailed documentation from SIKE [JAC+22] and SQIsign [AAA+25] for these building blocks, as the main contributions of QIMEN-PRISM lie in the protocol level. To clearly indicate reuse of material, we append a superscript $*$ to these sections.

Pseudocode conventions and exception handling

For clarity of exposition, this specification uses exceptions to describe error handling in pseudocode. An algorithm may raise an exception when a required randomized search or structural check fails. If a subroutine call raises an exception and the calling algorithm does not enclose that call in a **try**/**catch** block, the calling algorithm raises the same exception; equivalently, uncaught exceptions propagate to the next calling algorithm. When a **catch** block is present, the exception is handled as specified in that block.

Algorithm 1 Exception-handling convention

```
try
    Execute instructions that may raise an exception.
    if an error condition occurs then
        raise Exception("Description of the error")
catch
    Recover from the exception, derive an implicit-rejection fallback key, retry the randomized procedure, return false, or reject the input, depending on the algorithm.
```

We also use the standard loop-control convention that, inside a loop, the statement **continue** skips the remaining instructions of the current iteration and proceeds with the next iteration.

Table 1.3: Notation and parameters of the QIMEN-PRISM.

	Elementary Objects
$\mathbb{Z}/d\mathbb{Z}$	Ring of integers modulo d for some integer d.
$\mathbb{F}_p$	Finite fields with p elements.
$\mathbb{F}_q$	Finite fields with $q = p^k$ elements.
$\mathbb{F}_{p^2}$	Finite fields with p^2 elements.
$\overline{\mathbb{F}_p}$	Algebraic closure of $\mathbb{F}_p$.
$\mathrm{GL}_n(q)$	Group of invertible matrices of size n and elements in $\mathbb{F}_q$.
	Elliptic Curve Objects
E	An elliptic curve over some field $\mathbb{F}_q$.
$E \times E'$	A product of two elliptic curves E and E'.
A	An abelian variety over some field $\mathbb{F}_q$.
$\mathrm{Jac}(C)$	The Jacobian of a hyperelliptic curve C.
$E_{A,B}$	An elliptic curve in Montgomery form with parameters $A, B \in \mathbb{F}_q$.
0_E	The point at infinity of an elliptic curve or abelian variety E.
$E(\mathbb{F}_q)$	The points on E with coordinates in $\mathbb{F}_q$.
$E[n]$	The n-torsion on E.
$hintgen_i$	Hints to generate a basis for $E[2^f]$.
$j(E)$	The j-invariant of the elliptic curve E.
x_P and y_P	The x- resp. y-coordinate of a point $P \in E$.
$\varphi : E \to E'$	An isogeny between elliptic curves.
$\Phi : A \to A'$	A higher-dimensional isogeny.
$\mathrm{End}(E)$	The endomorphism ring of an elliptic curve E.
$t_n(P, Q)$	The degree-n Tate pairing evaluated at $P, Q \in E[n]$.
$e_n(P, Q)$	The degree-n Weil pairing evaluated at $P, Q \in E[n]$.
	Quaternion Objects
$\mathcal{B}_{p,\infty}$	A quaternion algebra, ramified at p and ∞.
$\mathcal{B}^\star_{p,\infty}$	The space of linear functions $\mathcal{B}_{p,\infty} \to \mathbb{Q}$.
$\mathcal{O}$	An order in $\mathcal{B}_{p,\infty}$.
$\mathcal{O}_0$	The special extremal maximal order satisfying $\mathrm{End}(E_0) \cong \mathcal{O}_0$.
$\mathcal{O}_L(I)$, $\mathcal{O}_R(I)$	The left, resp. right, order of an ideal $I \subset \mathcal{O}$.
$\mathrm{tr}(\alpha)$	The trace of an element $\alpha \in \mathcal{B}_{p,\infty}$.
$\mathrm{nrd}(\alpha)$	The norm of an element $\alpha \in \mathcal{B}_{p,\infty}$.
$\mathrm{nrd}(I)$	The norm of an ideal $I \subset \mathcal{O}$.
$[I]^*$	The pullback by an ideal I.
$[I]_*$	The pushforward by an ideal I.

Table 1.4: Notation and parameters of the QIMEN-PRISM.

	Security Parameters
λ_c	Target classical security strength, in bits.
λ_q	Target quantum security strength, in bits.
	Field Parameters
p	Field characteristic, chosen as $p = f \cdot 2^e - 1$ in the parameter sets.
e	Largest integer such that $2^e \mid p + 1$; determines the available 2-power torsion.
f	Odd cofactor in $p = f \cdot 2^e - 1$.
a	Integer parameter satisfying $0 < a < e$; its concrete value is fixed by the parameter set.
	Public and Secret Key Objects
E_0	Starting curve with equation $y^2 = x^3 + x$.
(P_0, Q_0)	Basis of $E_0[2^{a+2}]$.
N_{sk}	Prime degree of the secret isogeny, chosen as the smallest prime larger than $2^{4\lambda_c}$.
I_{sk}	Secret ideal of norm N_{sk}.
ϕ_{sk}	Secret isogeny $E_0 \to E_{\mathsf{pk}}$ corresponding to I_{sk}.
E_{pk}	Public-key curve obtained as the codomain of ϕ_{sk}.
P_{pk} and Q_{pk}	Generators of $E_{\mathsf{pk}}(\mathbb{F}_{p^2})[2^{a+2}]$ recovered from $\mathsf{hint}_{\mathsf{pk}}$.
$\mathsf{hint}_{\mathsf{pk}}$	Hint encoding the deterministic torsion basis $(P_{\mathsf{pk}}, Q_{\mathsf{pk}})$ on E_{pk}; stored in the public key.
M_{sk}	Change-of-basis matrix sending $(\phi_{\mathsf{sk}}(P_0), \phi_{\mathsf{sk}}(Q_0))$ to $(P_{\mathsf{pk}}, Q_{\mathsf{pk}})$.
	Hashing and Challenge Objects
$\mathsf{H}_{\mathsf{PRISM}}$	Hash function mapping $(j(E), \mathsf{msg}, \mathsf{salt})$ to an odd integer of length $a - 1$ bits.
H_{a-2}	Auxiliary hash function mapping bit strings to integers in $[0, 2^{a-2})$.
$\mathsf{PrimalityTest}$	Primality test applied to the output of $\mathsf{H}_{\mathsf{PRISM}}$; it is instantiated by Miller–Rabin with $\mathrm{MR}_{\mathrm{iter}}$ iterations.
q	Prime output of $\mathsf{H}_{\mathsf{PRISM}}$, used to define the signing isogeny degree $q(2^a - q)$.
I_{chall}	Challenge ideal of norm $q(2^a - q)$, constructed during signing.
salt	Salt sampled during signing and included with the signature.
n_{salt}	Bit length of salt.
	Signature and Verification Objects
E_{sig}	Signature curve, the codomain of the signing isogeny.
σ	Signing isogeny $E_{\mathsf{pk}} \to E_{\mathsf{sig}}$ of degree $q(2^a - q)$.
P_{sig} and Q_{sig}	Points on E_{sig} used to represent the signing isogeny for verification.
M_{sig}	Change-of-basis matrix used in the encoded signature.
$\mathsf{hint}_{\mathrm{sig}}$	Hint used to recover a torsion basis on E_{sig}.
accept and reject	Verification outputs.

CHAPTER 2

Mathematical foundations*

2.1 Finite fields

We follow the presentation in [JAC+22] and [AAA+25]. A finite field is a finite set of elements equipped with addition and multiplication operations that satisfy the natural rules for arithmetic. In particular, addition and multiplication are closed, there exist additive resp. multiplicative neutral elements 0 resp. 1, and additive resp. multiplicative inverses of each element, except for the non-multiplicatively-invertible element 0.

Finite fields of cardinality q exist if and only if q is a prime power, i.e., $q = p^r$ for some prime number p and positive integer r. Such finite fields of cardinality q have a unique representation up to isomorphism, and are denoted by $\mathbb{F}_q$. We denote their multiplicative group, i.e., $\mathbb{F}_q \setminus \{0\}$, by $\mathbb{F}_q^\times$. For $q = p^r$, we call $\mathrm{char}(\mathbb{F}_q) = p$ the characteristic of $\mathbb{F}_q$. QIMEN-PRISM uses fields of special characteristic that satisfy $p \equiv 3 \pmod 4$, so that we can always represent elements of $\mathbb{F}_{p^2}$ as $a + bi$ with $a, b \in \mathbb{F}_p$. We give more details on the specific primes p in Chapter 5.

We refer to the union of all finite fields of characteristic p as the *algebraic closure* of $\mathbb{F}_p$, denoted $\overline{\mathbb{F}_p}$. This is a field in itself, though no longer finite. When L is a field containing a smaller field K, while preserving addition and multiplication, we say that L is a *field extension* of K. Common examples include $\mathbb{F}_{p^2}$ as an extension of $\mathbb{F}_p$, or more generally $\mathbb{F}_q$ of $\mathbb{F}_p$.

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

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

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

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

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

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

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

$$\sqrt{a} = a^{\frac{p+1}{4}} \pmod{p}. \tag{2.1}$$

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

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

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

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

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

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

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

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

Square root. Let $a \in \mathbb{F}_p$, then a is always a square in $\mathbb{F}_{p^2}$. If a is a square in $\mathbb{F}_p$, we define its square root in $\mathbb{F}_{p^2}$ as in Eq. (2.1); otherwise $-a$ is a square in $\mathbb{F}_p$, and we

Algorithm 2 NormalizedDlog(a, ζ_0, ζ_1)

Input: A positive integer a and two values $\zeta_0, \zeta_1 \in \mu_{2^a}$, with $a \leq e$ and ζ_0 of order 2^a
Output: The value $k \in [0, 2^a]$ such that $\zeta_1 = \zeta_0^k$

if $a = 1$ **then**
  return k such that $\zeta_1 = \zeta_0^k$ by exhaustive search
$a' \leftarrow \lfloor a/2 \rfloor$
$\zeta_0' \leftarrow \zeta_0^{2^{a-a'}}$
$\zeta_1' \leftarrow \zeta_1^{2^{a-a'}}$
$k' \leftarrow$ NormalizedDlog(a', ζ_0', ζ_1')
$\zeta_0'' \leftarrow \zeta_0^{2^{a'}}$
$\zeta_1'' \leftarrow \zeta_1 / \zeta_0^{k'}$
$k'' \leftarrow$ NormalizedDlog$(a - a', \zeta_0'', \zeta_1'')$
return $k = k' + 2^{a'} k''$

define $\sqrt{a} = \sqrt{-a} \cdot i$. Finally, let $a \in \mathbb{F}_{p^2} \backslash \mathbb{F}_p$ be a square in $\mathbb{F}_{p^2}$. We define a canonical square root of a as

$$\sqrt{a} = (-i)^{\frac{1-\chi}{2}} S(a+\delta), \text{ where } \delta = \sqrt{a_0^2 + a_1^2},\ S = \big(2(a_0+\delta)\big)^{\frac{p-3}{4}},\ \chi = \big(2(a_0+\delta)\big)^{\frac{p-1}{2}}. \tag{2.2}$$

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

$$a_0 + a_1 \cdot i < b_0 + b_1 \cdot i \quad \text{iff} \quad a_0 < b_0 \text{ or } (a_0 = b_0 \text{ and } a_1 < b_1). \tag{2.3}$$

2.1.3 Discrete logarithm in finite field

Given a finite-field element $\zeta \in \mathbb{F}_{p^2}^\times$ and a power ζ^k for an unknown $k \in \mathbb{Z}$, the discrete logarithm problem (DLP) asks to recover k. When the order of ζ is smooth enough, this problem can be solved efficiently.[1] Let μ_n denote the group of n-th roots of unity, that is,

$$\mu_n := \{\zeta \in \overline{\mathbb{F}}_p \mid \zeta^n = 1\}.$$

Several algorithms exist to solve such discrete logarithms in μ_{2^a}, all tracing back to Pohlig–Hellman [PH78], and in general one can solve discrete logarithms as long as the order is smooth. In our implementation, we use the iterative algorithm NormalizedDlog, shown in Algorithm 2, to compute this discrete logarithm between two elements using the Pohlig–Hellman algorithm. QIMEN-PRISM only applies this algorithm to the output of pairing computations, see Section 2.3.2.

[1] When the order of ζ is a large prime, this problem is suspected to be hard for classical computers, and underlies the security of traditional Diffie–Hellman cryptography.

2.2 Elliptic curves

In the following, we assume that $\mathbb{F}_q$ is a finite field with $\mathrm{char}(\mathbb{F}_q) > 3$. Every *elliptic curve* over $\mathbb{F}_q$ can be given in a short Weierstrass equation $y^2 = x^3 + ax + b$ with $a, b \in \mathbb{F}_q$, and we take this to be the definition of elliptic curves.

We recall here some key facts on elliptic curves in Montgomery form necessary for the implementation of QIMEN-PRISM. For an extensive review of Montgomery curves and their properties, see [CS18].

2.2.1 Montgomery curves

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

$$By^2 = x^3 + Ax^2 + x. \tag{2.4}$$

That is, it consists of the set of points $P = (x, y)$ that satisfy the curve equation for $x, y \in \overline{\mathbb{F}_q}$, and the point at infinity 0_E. We often write $E_{A,B}/\mathbb{F}_q$ to emphasize that the curve is defined over the field $\mathbb{F}_q$, and we write $E_{A,B}(\mathbb{F}_q)$ to refer to the set of points (x, y) with $x, y \in \mathbb{F}_q$, plus 0_E. When $B = 1$, we write simply E_A and we write E for a generic Montgomery curve. We call the coefficient A the *Montgomery coefficient*.

Two Montgomery curves E and E' are said to be *isomorphic over* $\mathbb{F}_q$ if there is a linear change of coordinates $(x, y) \mapsto (Dx + R, Cy)$, with $D, C \in \mathbb{F}_q^\times$ and $R \in \mathbb{F}_q$, mapping E to E'. Two Montgomery curves $E_{A,B}$ and $E_{A,B'}$ are isomorphic when B/B' is a square in $\mathbb{F}_q$, Let N be the cardinality of $E(\mathbb{F}_q)$, that is, the number of solutions to Eq. (2.4) plus the point 0_E. When $N \equiv 1 \bmod \mathrm{char}(\mathbb{F}_q)$, we say that $E_{A,B}$ is *supersingular*. We are only interested in supersingular curves defined over $\mathbb{F}_{p^2}$ with $p \equiv 3 \bmod 4$. In this case, any supersingular curve $E_A/\mathbb{F}_{p^2}$ with $B = 1$ has exactly $(p+1)^2$ points, whereas its quadratic twists $E_{A,\gamma}/\mathbb{F}_{p^2}$, where γ is an arbitrary quadratic non-residue in $\mathbb{F}_{p^2}$, have exactly $(p-1)^2$ points and are all isomorphic. We refer to a curve with $(p+1)^2$ points as *maximal*, and to curves with $(p-1)^2$ points as *minimal*.

2.2.2 The group law

In this section we define the addition operation on the set of points of a Montgomery curve, so that $E_A(\mathbb{F}_q)$ forms an abelian group. Under this addition law, 0_E is the identity element, and each point $P = (x, y)$ has a unique inverse $-P = (x, -y)$ such that $P + (-P) = 0_E$.

In what follows, for a point $P \neq 0_E$, we refer to its x-coordinate as x_P, and to its y-coordinate as y_P, i.e., $P = (x_P, y_P)$. Note that optimized implementations typically use projective coordinates $(X : Y : Z)$ with $x = X/Z$ and $y = Y/Z$ in order to avoid inversions in the point addition and isogeny formulas below (see, e.g., [CS18]). Furthermore, we mostly use x-only arithmetic and represent points only as $P = (X_P : Z_P)$, which means that points are only defined up to the sign of their y-coordinates.

2.2.2.1 Point addition

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

$$x_R = B\lambda^2 - (x_P + x_Q) - A, \tag{2.5}$$

and

$$y_R = \lambda(x_P - x_R) - y_P, \tag{2.6}$$

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

To double a point $P = (x_P, y_P)$, we use the same formulas with λ replaced with $(3x_P^2 + 2Ax_P + 1)/(2By_P)$. Furthermore, if $P = -P$, then $[2]P = P + P = P + (-P) = 0_E$. The point at infinity is the neutral element of the law, so $P + 0_E = 0_E + P = P$.

2.2.2.2 Scalar multiplication

Using the abelian group law, we can define a scalar multiplication $[k] : E \to E$ for $k \in \mathbb{Z}$: for a point $P \in E$ we denote $[k]P = P + P + \cdots + P$ (k copies of P). For negative k, we set $[k]P = -[|k|]P$. For $k = 0$, we set $[0]P = 0_E$. For efficiency, a scalar multiplication is usually performed as a sequence of point doublings and point additions. Using the Montgomery ladder (see, e.g., [CS18]), the number of elliptic curve point operations is logarithmic in k. For a point $P \in E$, we call the smallest positive integer m such that $[m]P = 0_E$ the *order* of P.

2.2.2.3 Point difference

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

$$r_\pm = \frac{B_{XZ} \pm \sqrt{B_{XZ}^2 - B_{ZZ}B_{XX}}}{B_{ZZ}}, \tag{2.7}$$

where

$$\begin{aligned} B_{XX} &= (x_P x_Q - 1)^2, \\ B_{XZ} &= (x_P x_Q + 1)(x_P + x_Q) + 2Ax_P x_Q, \\ B_{ZZ} &= (x_P - x_Q)^2. \end{aligned} \tag{2.8}$$

As $x_Q = x_{-Q}$, we cannot determine which of r_+ and r_- is x_{P-Q} and which is x_{P+Q} based on x-coordinates alone. However, in practice we are also free to replace Q by $-Q$ and so it suffices to deterministically choose one of the two. We make this choice by always using the formula for r_+, with the choice of square root described in Section 2.1.2. This routine is referred to as ProjectiveDifference, since in practice it is implemented in projective coordinates, see Section C.2.

Coefficient recovery. Given the x-coordinates x_P and x_Q as well as the coordinate x_R of $R = P - Q$ of three points $P, Q, R \in E_A$, we are able to recover the Montgomery coefficient A by rewriting Equation (2.8). The resulting algorithm is given in RecoverCodomain, see Section C.2.

2.2.3 Torsion subgroups and deterministic basis computation

For $m \in \mathbb{Z}$ and $E/\mathbb{F}_{p^2}$ a supersingular curve, we define $E[m]$ to be the m-torsion subgroup of E, which contains all points $P \in E(\overline{\mathbb{F}_p})$ such that $[m]P = 0_E$. If $m \nmid p$, we have $E[m] \cong \mathbb{Z}/m\mathbb{Z} \times \mathbb{Z}/m\mathbb{Z}$. For a maximal supersingular curve, when $m \mid p+1$ then $E[m] \subseteq E(\mathbb{F}_{p^2})$. Thus, there are non-unique points $R, S \in E[m]$ that generate $E[m]$. We refer to such a pair of points as a *basis* of $E[m]$, or similarly, we write $E[m] = \langle R, S \rangle$. In x-only coordinates, we may represent such a basis (R, S) by x_R, x_S, and the x-coordinate of either $R + S$ or $R - S$, which we denote by X_{RS}. Thus, the reference implementation represents a basis (R, S) by a triplet (x_R, x_S, x_{RS}). Technically, this defines (R, S) only up to a global sign, i.e., the representation is the same for (R, S) and $(-R, -S)$.

Throughout this document, e denotes the largest integer such that 2^e divides $p+1$, for p the base prime of the finite field. In QIMEN-PRISM, we need to generate a basis (R, S) for $E[2^a]$ for $a \leq e$, which is well-studied [ZSP+18, CEMR24]. We describe a deterministic generation of a basis (R, Q) for $E[2^a]$ in TorsionBasisToHint, using a modified version of [ZSP+18, Alg. 3.1] for Montgomery curves E_A with $A \neq 0$ over $\mathbb{F}_{p^2}$: This algorithm computes the quadratic residuosity of A to ensure that $x_R \leftarrow -A/(1 + i \cdot b)$ is a non-square in $\mathbb{F}_{p^2}$, for a square b. Then, if $x_R \in E(\mathbb{F}_{p^2})$, this ensures that $x_S = -x_R - A$ is also a non-square, and that together (R, S) is a basis of $E[2^a]$, when multiplied by the cofactor $(p+1)/2^a$. We adjust this algorithm by using that $b \in \mathbb{F}_p$ is always a square in $\mathbb{F}_{p^2}$, hence when A is non-square, any $\mathbb{F}_p$-multiple $b \cdot A$ is of the right form for x_R. One can show that for a basis (R, S) sampled using this algorithm, the point $R + S$ satisfies $2^{a-1}(R + S) = (0, 0)$, which is a property we assume in later algorithms. As we want this property for S instead of $R+S$, we use $(R, R+S)$ as our basis by permuting (x_R, x_S, x_{RS}) to (x_R, x_{RS}, x_S). Computing an x_{RS} given x_R and x_S must be done deterministically, however, it is impossible to know given only x_R and x_S if x_{RS} is the x-coordinate of $R+S$ or $R-S$. This is not an issue, as long as an implementation makes the same deterministic choice as the reference implementation, which uses ProjectiveDifference, see Section C.2. For the particular elliptic curve E_0 for which this algorithm does not apply, we may precompute a basis of $E_0[2^e]$ during parameter generation, so this does not require careful optimization.

Hints. Sampling such a torsion basis for $E_A[2^e]$ 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_P. Both of these can be provided in the signature as hints: the signer uses TorsionBasisToHint, which generates a basis and the corresponding hints, and includes the hints in the signature. The verifier uses TorsionBasisFromHint, which takes the hints

Algorithm 3 TorsionBasisToHint(A, a)

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

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

as input and returns the basis (x_R, x_S, x_{RS}). Note that a basis returned using TorsionBasisFromHint is guaranteed to have two values x_R and x_S on the same twist, even when these values are not verified as points on E. This causes no problems, as long as the orders of these points are verified whenever they are used, which implicitly ensures they lie on E. The reference implementation does this as described in Chapters C and D.

2.3 Pairings

This section introduces the Weil and Tate–Lichtenbaum pairings of level n, which we use to efficiently compute discrete logarithms between points and for torsion basis generation: Pairings allow us to translate the elliptic-curve discrete logarithm problem into a finite-field discrete logarithm problem, which can then be solved efficiently with a function like NormalizedDlog whenever the order is smooth. This approach is particularly attractive

Algorithm 4 TorsionBasisFromHint(A, a, h_A, h)

Input: A non-zero affine Montgomery coefficient A, an integer $a < e$, together with two hints h_A, h

Output: An x-only basis (x_R, x_S, x_{RS}) of $E[2^a]$

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

in the context of QIMEN-PRISM, as we are primarily concerned with discrete logarithm computations of points belonging to $E[2^a]$, where $a \leq e$. As $E[2^e] \subseteq E(\mathbb{F}_q)$ for the curves we work with, such pairing computations are efficient.

2.3.1 Pairings on elliptic curves

Pairings are bilinear maps $A \times B \to C$ between abelian groups A, B, and C. Most relevant for cryptography are the Weil and Tate–Lichtenbaum pairings of level n, where A and B are subgroups or quotient groups of $E[n]$ and C is the group of n-th roots of unity $\mu_n \subset \mathbb{F}_{p^2}^\times$.

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

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

which is bilinear, alternating, and non-degenerate.

The Tate–Lichtenbaum pairing. The Tate–Lichtenbaum pairing is a pairing defined by Tate [Tat62] for abelian varieties over local fields, with Lichtenbaum [Lic69] showing its efficient computation for Jacobians of curves. Frey and Rück [FR94] showed its cryptographic application, including an efficient algorithm over finite fields. Assume $n \mid q - 1$. We define the unreduced Tate–Lichtenbaum pairing as

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

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

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

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

2.3.2 Use of pairings in QIMEN-PRISM

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

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

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

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

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

There are several methods to compute t_n efficiently; see Chapter E for the choice made in QIMEN-PRISM and details on these algorithms.

The Weil pairing. In QIMEN-PRISM, the Weil pairing is used in both signing and verification to verify the degree of an isogeny. This uses the property of the Weil pairing that $e_n(\phi P,\phi Q) = e_n(P,Q)^{\deg\phi}$ whenever $\deg\phi$ is coprime with n. We apply this for $n = 2^a$ by computing both $\zeta = e_{2^a}(P,Q)$ and $\zeta' = e_{2^a}(\phi P,\phi Q)$ for a basis (P,Q) of the 2^a-torsion, which implies that ζ is a 2^a-th root of unity. Verifying that $\zeta' = \zeta^m$ then shows that $\deg\phi = m \mod 2^a$.

Algorithm 5 ChangeOfBasis$_{2^a}(E, (P_1, P_2), (Q_1, Q_2))$

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

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

In signing, this is used in IdealToIso to identify the correct codomain, whereas in verification, this is used in CheckIsogeny to verify the degree of an isogeny ϕ.

The Weil pairing can be computed efficiently as a ratio of two Tate pairings; see Chapter E for the choice made in QIMEN-PRISM and details on these algorithms.

2.4 1-Dimensional isogenies

Remark 2.4.1. *QIMEN-PRISM does not explicitly use or compute 1-dimensional isogenies, and so we leave out such details in this section. However, their general description is very useful to grasp 2-dimensional isogenies, which we introduce in Section 2.5.*

For two elliptic curves E_1 and E_2 over $\mathbb{F}_q$, an *isogeny* is a non-constant map $\varphi : E_1 \to E_2$ defined coordinate-wise by polynomial fractions over $\mathbb{F}_q$, that satisfies $\varphi(0_{E_1}) = 0_{E_2}$. In particular, φ is a group homomorphism $\varphi : E_1 \to E_2$. Such curves E_1 and E_2 that are connected through an isogeny are called isogenous. A characterization for this property is given by the group orders: two curves $E_1/\mathbb{F}_q$ and $E_2/\mathbb{F}_q$ are isogenous over $\mathbb{F}_q$ if and only if $\#E_1(\mathbb{F}_q) = \#E_2(\mathbb{F}_q)$. We denote composition of isogenies by $\circ$.

Every isogeny φ has a finite *degree*, denoted by $\deg(\varphi)$. Concretely, φ induces a pullback map on rational functions $\varphi^* : \mathbb{F}_q(E_2) \hookrightarrow \mathbb{F}_q(E_1)$, and $\deg(\varphi)$ is the degree of the field extension $[\mathbb{F}_q(E_1) : \varphi^*\mathbb{F}_q(E_2)]$. Its kernel is the finite subgroup of geometric points $\ker(\varphi) = \{P \in E_1(\overline{\mathbb{F}}_q) \mid \varphi(P) = 0_{E_2}\}$. The isogeny φ is called *separable* if $\#\ker(\varphi)(\overline{\mathbb{F}}_q) = \deg(\varphi)$. For supersingular curves, this is the case precisely for isogenies whose degree is coprime to p.

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

a coprime degree, we may push the kernel of one isogeny, say $K = \ker \psi$ forward through the other isogeny to get a third isogeny $\psi' : E_1 \to E_3$ with kernel $\phi(K)$. We call this the *pushforward* of ψ by ϕ, and denote this by $[\phi_*]\psi$.

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

2.5 2-Dimensional isogenies

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

Principally polarized abelian surfaces (PPAS) are a natural generalization of elliptic curves (see Section 2.2) to two dimensions. In particular, PPAS are geometric objects defined by polynomial equations to which we can associate a group whose group law is given by rational functions, i.e., fractions of polynomials. Over an algebraically closed field such as $\overline{\mathbb{F}_p}$, PPAS are isomorphic to either one of the following [Wei57]:

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

The arithmetic on the Cartesian product $E_1 \times E_2$ of elliptic curves E_1, E_2 follows immediately from the arithmetic on the curves E_1 and E_2 themselves: addition $(P_1, P_2), (Q_1, Q_2) \in E_1 \times E_2$ is defined as

$$(P_1, P_2) + (Q_1, Q_2) := (P_1 + Q_1,\ P_2 + Q_2), \tag{2.9}$$

where the addition of P_1 and Q_1 happens on E_1 and the addition of P_2 and Q_2 happens on E_2. On $E_1 \times E_2$, the neutral element is $(0_{E_1}, 0_{E_2})$.

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

$$C : y^2 = f(x), \tag{2.10}$$

where f is squarefree polynomial over $\mathbb{F}_{p^2}$ with $\deg(f) = 5$ or 6, when $p > 5$. Unlike elliptic curves, the curve C does not form a group under point addition. Instead, we construct the Jacobian $\mathrm{Jac}(C)$ [Mil86], which is an abelian group associated to the curve C. Note that in the case of an elliptic curve E, we have $\mathrm{Jac}(E) \simeq E$, which is why the construction of the Jacobian can usually be avoided. The group law on $\mathrm{Jac}(C)$ can be computed using Cantor's algorithm [Can89]. We denote the neutral element as 0_J, or 0_A when we talk about a general PPAS A. Henceforth, when we write $\mathrm{Jac}(C)$, we always mean the Jacobian of a genus-2 hyperelliptic curve C.

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

- An isogeny $\Phi : E_1 \times E_2 \to \mathrm{Jac}(C)$ is called a *gluing isogeny*,
- An isogeny $\Phi : \mathrm{Jac}(C) \to E_1 \times E_2$ is called a *splitting isogeny*,
- An isogeny $\Phi : \mathrm{Jac}(C) \to \mathrm{Jac}(C')$ is called a *generic isogeny*,
- Otherwise, an isogeny $\Phi : E_1 \times E_2 \to E_3 \times E_4$ of the form $\Phi(P, Q) = (\phi_1(P), \phi_2(Q))$, where $\phi_1 : E_1 \to E_3$ and $\phi_2 : E_2 \to E_4$, is called a *diagonal isogeny*.

Similar to elliptic curve isogenies, isogenies between PPAS have a finite kernel

$$\ker(\Phi) = \{P \in A_1 \mid \Phi(P) = 0_{A_2}\}$$

and are determined by their kernel up to post-composition with an isomorphism. We say that Φ is an (N, N)-isogeny when its kernel $\ker(\Phi)$ is generated by two linearly independent points $P, Q \in A_1$ of order N so that $\ker(\Phi) \cong \mathbb{Z}/N\mathbb{Z} \times \mathbb{Z}/N\mathbb{Z}$. In this context, linearly independent means that for all integers k and l, we have $[k]P + [l]Q = 0_{A_1}$ if and only if $N \mid k$ and $N \mid l$. We write $\ker(\Phi) = \langle P \rangle \oplus \langle Q \rangle$. The isogeny Φ can then be represented by P and Q and computed from these points with generalizations of Vélu's formulas in time $O(N^2)$ [LR23].

Not all choices of linearly independent N-torsion points P, Q define an isogeny. To define an isogeny of PPAS, it is necessary and sufficient that P and Q are *isotropic*, i.e., P, Q have trivial Weil pairing[2]: $e_N(P, Q) = 1$. When N has prime decomposition $N = \prod \ell_i^{e_i}$, rather than computing an (N, N)-isogeny Φ with complexity $O(N^2)$, we may decompose Φ as

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

where the Φ_i are (ℓ_i, ℓ_i)-isogenies. The isogeny Φ can now be computed more efficiently in $O(\sum e_i \ell_i^2)$.

In QIMEN-PRISM. For QIMEN-PRISM, we specialize to the case of $(2^k, 2^k)$-isogenies

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

between elliptic curve products $E_1 \times E_2$ and $E_3 \times E_4$ defined over $\mathbb{F}_{p^2}$, that we decompose into a chain of $(2, 2)$-isogenies of length k, for some integer k. In QIMEN-PRISM, we expect the first isogeny of the chain $\Phi_1 : E_1 \times E_2 \to A_1$ to be a gluing isogeny, the last step $\Phi_k : A_{k-1} \to E_3 \times E_4$ to be a splitting isogeny, and all intermediate isogenies to be generic.

[2]Similar to elliptic curves, we can define the Weil pairing on all PPAS.

2.5.1 Implementation details for $(2,2)$-isogenies

In Chapter D, we give algorithmic details on how $(2,2)$-isogenies are computed. For practical purposes, we use *theta coordinates of level* 2 to represent points on a PPAS, which can be thought of as a higher-dimensional generalization of x-only arithmetic on elliptic curves (see Section 2.2.2). Working with theta coordinates in isogeny-based cryptography is well-established. Due to their rather technical nature, we defer the details of these algorithms to Chapter D, and only highlight the main algorithms here.

- ThetaDBL: given theta coordinates of a point P and the constants `consts`, outputs theta coordinates for the point $[2]P$.
- GenericCodomainWith8Torsion: given theta coordinates of 8-torsion points T_1'', T_2'' on domain surface A, this algorithm outputs the dual theta null point $(\alpha : \beta : \gamma : \delta)$, its inverse $(\alpha^{-1} : \beta^{-1} : \gamma^{-1} : \delta^{-1})$, and the theta null point 0_B of the image B of the isogeny $\Phi : A \to B$ with $\ker(\Phi) = \langle [4]T_1'' \rangle \oplus \langle [4]T_2'' \rangle$.
- GenericCodomainWith4Torsion: given theta coordinates of a 4-torsion point T_1' on domain surface A satisfying $[2]T_1' \in \ker(\Phi)$, and the theta null point 0_A of A, this algorithm outputs the dual theta null point $(\alpha : \beta : \gamma : \delta)$, its inverse $(\alpha^{-1} : \beta^{-1} : \gamma^{-1} : \delta^{-1})$, and the theta null point 0_B of the image B of $\Phi : A \to B$.
- GenericCodomain: given the theta null point 0_A of domain A, computes the dual theta null point $(\alpha : \beta : \gamma : \delta)$, its inverse $(\alpha^{-1} : \beta^{-1} : \gamma^{-1} : \delta^{-1})$, and the theta null point 0_B of the image B of a $(2,2)$-isogeny $\Phi : A \to B$.
- GenericEval: given a point P (in theta coordinates) on domain surface A and the point $(\alpha^{-1} : \beta^{-1} : \gamma^{-1} : \delta^{-1})$, outputs the theta coordinates of $\Phi(P)$.
- GluingCodomain: given theta coordinates of 8-torsion points T_1'', T_2'' on domain product surface, this algorithm outputs the dual theta null point $(\alpha : \beta : \gamma : 0)$, its "inverse" $(\alpha^{-1} : \beta^{-1} : \gamma^{-1} : 0)$, the theta null point 0_B of the image surface A of the isogeny $\Phi : E_1 \times E_2 \to A$ with $\ker(\Phi) = \langle [4]T_1'' \rangle \oplus \langle [4]T_2'' \rangle$, the dual of the theta point $\Phi(T_1'')$ and a change-of-basis matrix N (re-used for evaluation).
- GluingEval: given a point $P \in E_1 \times E_2$, T_1'' an 8-torsion point such that $[4]T_1'' \in \ker(\Phi)$, the dual theta point $\Phi(T_1'')$ on A, and the change-of-basis matrix N computed during GluingCodomain, outputs the theta coordinates of $\Phi(P)$.
- GluingEvalSpecial: given a point $P \in E_1 \times E_2$ of the form $(P_1, 0)$ or $(0, P_2)$, $(\alpha^{-1} : \beta^{-1} : \gamma^{-1} : 0)$ the "inverse" of the dual theta null point over A, and the change-of-basis matrix N computed during GluingCodomain, outputs the theta coordinates of $\Phi(P)$.
- SplittingIsomorphism: given the theta null point 0_A on surface $A \cong E_1 \times E_2$, computes the isomorphism whose action on 0_A gives the theta null point associated with the product theta structure.

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

Using the core algorithms from the previous section, we can describe the computation of a chain of $(2,2)$-isogenies. Consider a $(2^a, 2^a)$-isogeny $\Phi : E_1 \times E_2 \to E_3 \times E_4$ between products of elliptic curves. Knowing two points P and Q such that $\ker(\Phi) = \langle [4]P, [4]Q \rangle$, we explain how to compute Φ as a chain of $(2,2)$-isogenies:

$$E_1 \times E_2 \xrightarrow{\Phi_1} A_1 \xrightarrow{\Phi_2} A_2 \cdots \xrightarrow{\Phi_{a-1}} A_{a-1} \xrightarrow{\Phi_a} E_3 \times E_4.$$

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

1. For Φ_1: Using GluingCodomain, compute the gluing isogeny $\Phi_1 : E_1 \times E_2 \to A_1$ using the 8-torsion points $[2^a]P$ and $[2^a]Q$ lying above $\ker(\Phi_1) = \langle [2^{a+1}]P, [2^{a+1}]Q \rangle$. Then, using GluingEval, compute $\Phi_1(P), \Phi_1(Q)$.

2. For Φ_i with $2 \leq a$: Using GenericCodomainWith8Torsion, compute the generic isogeny $\Phi_i : A_{i-1} \to A_i$ using the 8-torsion points $[2^{a-i}](\Phi_{i-1} \circ \cdots \circ \Phi_1(P))$ and $[2^{a-i}](\Phi_{i-1} \circ \cdots \circ \Phi_1(Q))$ lying above $\ker(\Phi_i)$. Note that in the last step, we obtain a theta null point for $A_a = E_3 \times E_4$, and not yet these curves themselves.

3. For $E_3 \times E_4$: Using SplittingIsomorphism, compute a change of coordinates from the system of theta coordinates on $E_3 \times E_4$ obtained from Φ_a to a system of product theta coordinates in order to express image points on $E_3 \times E_4$ in $(X : Z)$-Montgomery coordinates on each component E_3 and E_4.

By starting with points P and Q of order 2^{a+2} such that $\ker \Phi = \langle [4]P, [4]Q \rangle$, we avoid costly square root computations in as we may use GenericCodomainWith8Torsion for every Φ_i with $2 \leq i \leq a$, including the last two steps, using the 8-torsion points $[2^{a-i}](\Phi_{i-1} \circ \cdots \circ \Phi_1(P))$ and $[2^{a-i}](\Phi_{i-1} \circ \cdots \circ \Phi_1(Q))$. This optimization is only possible when 2^{a+2}-torsion points are defined over $\mathbb{F}_{p^2}$, which is not always the case. In QIMEN-PRISM, the degree a is chosen so that $a + 2 \leq e$, where 2^e is the maximal available $2^\bullet$-torsion, so we always choose this approach.

Strategies. The above description to compute Φ is called the *naive strategy*, which is unoptimal. Instead, we use a *balanced strategy*: By storing intermediate points obtained during the doublings and pushing them through each isogeny, we can reduce the number of executions of the doubling algorithm ThetaDBL to a quasi-linear number $O(a \log(a))$.[3]

[3] We choose not to use *optimal strategies*, as used in SIDH/SIKE [JD11, § 4.2.2], as they give only a small efficiency gain, but have a moderate memory cost due to the need to store the (precomputed) strategies.

2.5.3 Implementation details for chains of isogenies

In Chapter D, we present the algorithmic details for the full computation of a $(2^a, 2^a)$-isogeny, as a single algorithm that encompasses all of the steps in Section 2.5.2. As we always assume $a + 2 \leq e$, we write only give one approach:

- Isogeny22Chain: on input isotropic points $P, Q \in E_1 \times E_2$ of order 2^{a+2} with $a + 2 \leq e$, and an array `pts` containing points on $E_1 \times E_2$, outputs a $(2, 2)$-isogeny chain $\Phi = \Phi_a \circ \cdots \circ \Phi_1$ such that $\ker(\Phi) = \langle [4]P, [4]Q \rangle$, and evaluated points $\{\Phi(R) : R \in \texttt{pts}\}$, computed using balanced strategies.

CHAPTER 3

Quaternions

This section introduces quaternions, an essential algebraic object that allows the construction of QIMEN-PRISM. In Section 3.1, we discuss some basic arithmetic of integers, matrices and lattices. In Section 3.2, we define quaternion algebra, quaternion orders, quaternion ideals, and introduce an algorithm that solves norm equations in a quaternion order. Finally in Section 3.3, we explain the connections between ideals and isogenies, and then provide an algorithm that turns a quaternion ideal into an isogeny efficiently.

3.1 Integers, matrices and lattices*

3.1.1 Integer arithmetic

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

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

- Basic arithmetic (addition, multiplication, ...) of integers;
- Uniform sampling of integers from an interval;
- Approximate and exact integer square roots;
- Pseudo-primality testing using the Miller–Rabin test;
- Extended greatest common divisor XGCD: given (a, b), find integers (g, u, v) such that $ua + bv = g = \gcd(a, b) > 0$ and if both $a \neq 0$ and $b \neq 0$, then $1 \leq au \leq |ab|/g$ and $-|ab|/g < bv \leq 0$ (the XGCD algorithm of GMP does however not enforce that $u \neq 0$, which is required in QIMEN-PRISM);
- Arithmetic modulo integers;

- Legendre symbol;
- Dyadic valuation of an integer;
- Square roots modulo primes.

With the exception of modular square roots (for which pseudocode is given in ModularSQRT), all these algorithms are implemented in GMP, which is the big integer library used by the QIMEN-PRISM reference implementation.

Algorithm 6 ModularSQRT(n, m)

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

if $n \equiv 0 \pmod{m}$ **then return** 0
if $m \equiv 3 \pmod{4}$ **then return** $n^{(m+1)/4} \bmod m$
if $m \equiv 5 \pmod{8}$ **then**
    if $n^{(m-1)/4} \equiv 1 \pmod{m}$ **then return** $n^{(m+3)/8} \bmod m$
    elsereturn $2n(4n)^{(m-5)/8} \bmod m$
$w \leftarrow 2$ and $e \leftarrow \mathsf{DyadicValuation}(m-1)$
$q \leftarrow (m-1)/2^e$
while w is a square modulo m **do**
    $w \leftarrow w + 1$
$z \leftarrow w^q \bmod m$ and $r \leftarrow e$ and $y \leftarrow n^q \bmod m$
$x \leftarrow n^{(q+1)/2} \bmod m$ and $f \leftarrow 2^{e-2}$
for i from 0 up to $e-1$ **do**
    $b \leftarrow y^f \bmod m$
    if $b = m - 1$ **then**
        $x \leftarrow xz \bmod m$
        $y \leftarrow yz^2 \bmod m$
        $z \leftarrow z^2 \bmod m$
        $f \leftarrow f/2$
    return x

3.1.2 Matrix arithmetic and Hermite normal form

Basic integral matrix arithmetic QIMEN-PRISM needs to manipulate several integer matrices of small dimensions (such as 2×2, 4×4, $4\times n$ for $n \leq 16$). Basic operations such as matrix-vector and matrix-matrix multiplication can be implemented using the schoolbook method. For determinants and inversion of 2×2 matrices the standard formulas can be used:

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

For determinants and inversion of 4×4 matrices a similar Laplacian formula [Ebe07] computes the determinant and the adjugate matrix using 78 ring operations.

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

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

A matrix A is said to have H for HNF if H is in HNF and there exists a unimodular matrix U such that $AU = H$, here unimodular means that U has integer coefficients and determinant ± 1. Then A and H have the same column space, and H is unique, thus giving a canonical representation for A's column space. An algorithm for computing the HNF of arbitrary matrices is presented in [Coh93, 2.4.2]. We provide an example of an algorithm that computes the Hermite normal form of a matrix in HNF.

In QIMEN-PRISM, ideals are represented by their basis, which is a 4×4 matrix whose columns are the basis vectors. The HNF of this matrix is used as a canonical representation of the ideal, and it is also used to check for equality of ideals, to compute their sum and intersection, and to compute the index of one ideal in another. See Section 3.2 and Section 3.2.2 for more details.

3.1.3 Lattice and lattice reduction

Let V be a d-dimensional $\mathbb{Q}$-vector space. A full rank lattice $\mathcal{L} \subseteq V$ is the $\mathbb{Z}$-span a $\mathbb{Q}$-basis of V, i.e., $\mathcal{L} = \mathbb{Z}b_0 + \cdots + \mathbb{Z}b_{d-1}$ for a basis $\{b_0, \ldots, b_{d-1}\}$ of V. Since we will only encounter full rank lattices in this document, we omit "full rank" when we refer to them.

The lattice we consider in QIMEN-PRISM will live in a *quadratic space*, which is a finite-dimensional vector space equipped with a symmetric bilinear form as defined below.

Definition 3.1.1 (Symmetric bilinear form)**.** A *symmetric bilinear form* on a $\mathbb{Q}$-vector space V is a mapping that to any $a, b \in V$ associates a value $\langle a, b \rangle \in \mathbb{Q}$ with the properties

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

for any $a, b \in V$ and any $\lambda \in \mathbb{Q}$. Two vectors $a, b \in V$ are said to be *orthogonal* if $\langle a, b \rangle = 0$.

To any symmetric bilinear form is associated a *quadratic form* defined by $Q(a) = \langle a, a\rangle$. Conversely, the bilinear form can be recovered from the quadratic form by the formula $\langle a, b\rangle = (Q(a+b) - Q(a) - Q(b))/2$.

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

Given a basis $b_0, \ldots, b_{d-1}$ of a quadratic space V, its *Gram matrix* is the symmetric matrix whose (i, j)-th entry is $\langle b_i, b_j\rangle$. This matrix uniquely determines the bilinear form on the lattice spanned by $b_0, \ldots, b_{d-1}$. Lattices of quadratic spaces do not always admit an orthogonal basis. The goal of *lattice reduction* is to compute a basis of a lattice that is "as orthogonal as possible" in a precise sense.

Definition 3.1.2 (Reduced basis)**.** Let $b_0, \ldots, b_{d-1}$ be a basis of a positive definite quadratic space. Define the Gram–Schmidt vectors $\bar{b}_i$ as

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

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

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

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

We will see the symmetric bilinear forms we consider in QIMEN-PRISM in Section 3.2.1. In QIMEN-PRISM, in order to use the IdealToIso algorithm for both key generation and signing, a crucial step is to compute a reduced basis of the input lattice. In a Euclidean space where the bilinear form is given by the inner product of $\mathbb{Q}^n$, this step would be done by the LLL algorithm [LLL82]. In the case of QIMEN-PRISM, where the bilinear form is no longer the standard one, we adopt the lattice reduction algorithm $\mathrm{L2}_{\eta,\delta}$ from [NS09]. We provide details of this algorithm in Section F.1.

Floating point QIMEN-PRISM makes a limited use of floating-point numbers in the implementation of the lattice reduction algorithm $\mathrm{L2}_{\eta,\delta}$ (see Section F.1). The native floating-point types of most platforms are insufficient for QIMEN-PRISM s needs, however any type with at least 24 bits of mantissa and at least 20 bits of exponent is largely sufficient for all security levels.

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

The basis output by a given lattice reduction algorithm implemented using floating-point numbers is highly sensitive to many factors, such as the exact order of operations performed (due to the non-associativity of floating-point numbers), the rounding mode in use, and possibly even the choice of compiler flags. Therefore, it may prove challenging for an alternative implementation of QIMEN-PRISM to exactly reproduce the Known Answer Tests described in Chapter 6; however, this does not preclude the generation of valid signatures, which are verified by a correct implementation of Algorithm 21.

3.2 Quaternions and ideals*

3.2.1 Basic definitions

Let $p \equiv 3 \bmod 4$ be a prime. The quaternion algebra $\mathcal{B}_{p,\infty}$ is the 4-dimensional $\mathbb{Q}$ vector space generated by four elements $\{1, \mathbf{i}, \mathbf{j}, \mathbf{k}\}$ such that

$$\mathbf{i}^2 = -1, \qquad \mathbf{j}^2 = -p, \qquad \mathbf{ij} = -\mathbf{ji} = \mathbf{k}. \tag{3.1}$$

This vector space becomes a $\mathbb{Q}$-algebra with

- $x \cdot (a + b\mathbf{i} + c\mathbf{j} + d\mathbf{k}) := (xa) + (xb)\mathbf{i} + (xc)\mathbf{j} + (xd)\mathbf{k}$ where $x, a, b, c, d \in \mathbb{Q}$;
- the multiplication of two elements in $\mathcal{B}_{p,\infty}$ $(a+b\mathbf{i}+c\mathbf{j}+d\mathbf{k})(a'+b'\mathbf{i}+c'\mathbf{j}+d'\mathbf{k})$ follows from the distributive law of this multiplication together with Eq. (3.1).

An element α of $\mathcal{B}_{p,\infty}$ is represented as a 4-tuple of rational numbers $(a, b, c, d) \in \mathbb{Q}^4$, representing

$$a + b\mathbf{i} + c\mathbf{j} + d\mathbf{k}.$$

Hence we abuse notation and use α to refer to either this quaternion element, or its representation by a vector $(a, b, c, d)^t \in \mathbb{Q}^4$. In the actual implementation, this is stored as a 5-tuple in $\mathbb{Z}^5$, where a canonical representation is obtained by reducing the common denominator.

Addition and **multiplication** of elements in $\mathcal{B}_{p,\infty}$ are as explained above. We further define other operations on $\mathcal{B}_{p,\infty}$ as follows: Let $\alpha = a + b\mathbf{i} + c\mathbf{j} + d\mathbf{k} \in \mathcal{B}_{p,\infty}$,

Conjugation: The conjugate $\bar{\alpha}$ of α is the element $\bar{\alpha} := a - b\mathbf{i} - c\mathbf{j} - d\mathbf{k}$.

Reduced trace: The reduced trace of α is $\mathrm{tr}(\alpha) := \alpha + \bar{\alpha} = 2a$.

Reduced norm: The reduced norm of α is $\mathrm{nrd}(\alpha) := \alpha\bar{\alpha} = a^2 + b^2 + p(c^2 + d^2)$.

The map

$$\langle \alpha, \beta \rangle = \mathrm{tr}(\alpha\bar{\beta}) \tag{3.2}$$

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

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

implying that the quadratic space is positive definite.

3.2.2 Quaternion lattices.

A *full rank lattice* $\Lambda \subseteq \mathcal{B}_{p,\infty}$ is defined by a basis $(\alpha_1, \alpha_2, \alpha_3, \alpha_4)$ of $\mathbb{Q}$-linearly independent quaternions. Following the convention on representing quaternion elements, a basis is then represented as *columns* of a matrix L, so that

$$(\alpha_1\ \alpha_2\ \alpha_3\ \alpha_4) = (1\ \mathbf{i}\ \mathbf{j}\ \mathbf{k}) \cdot L.$$

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

The *dual* of a lattice Λ is defined as

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

where $\mathcal{B}^\star_{p,\infty}$ denotes the space of linear functions $B_{p,\infty} \to \mathbb{Q}$. Let $(1^\star, \mathbf{i}^\star, \mathbf{j}^\star, \mathbf{k}^\star)$ be the *dual basis* to $(1, \mathbf{i}, \mathbf{j}, \mathbf{k})$, i.e., for $\alpha = a + b\mathbf{i} + c\mathbf{j} + d\mathbf{k}$,

$$1^\star(\alpha) = a, \qquad \mathbf{i}^\star(\alpha) = b, \qquad \mathbf{j}^\star(\alpha) = c, \qquad \mathbf{k}^\star(\alpha) = d.$$

This is the basis of the dual lattice $(\mathbb{Z} + \mathbf{i}\mathbb{Z} + \mathbf{j}\mathbb{Z} + \mathbf{k}\mathbb{Z})^\star$. If Λ is generated by a basis $(1\ \mathbf{i}\ \mathbf{j}\ \mathbf{k}) \cdot L$, then $\Lambda^\star$ is generated by $L^{-1} \cdot (1^\star\ \mathbf{i}^\star\ \mathbf{j}^\star\ \mathbf{k}^\star)^t$.

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

Equality (Algorithm 67): Check that L_1 and L_2 have the same HNF [Coh93, 2.4.3].

Sum (Algorithm 68): If L_1 and L_2 are lattices, concatenate their matrices $L_1 \mid L_2$ and compute the HNF to obtain $L_1 + L_2$.

Intersection (Algorithm 70): If L_1 and L_2 are lattices, compute their dual lattices $L_1^\star$ and $L_2^\star$; then $L_1 \cap L_2$ is the dual of $L_1^\star + L_2^\star$.

Multiplication (Algorithm 71): If L_1 and L_2 are lattices, their product $L_1 L_2$ is computed, e.g., by writing the right multiplication matrices $A_1, \ldots, A_4$ of a basis $\alpha_1, \ldots, \alpha_4$ of L_2, and then computing the sum

$$A_1 L_1 + A_2 L_1 + A_3 L_1 + A_4 L_1.$$

Containment (Algorithm 72): Given an element $\alpha \in \mathcal{B}_{p,\infty}$ and a lattice L, checking whether $\alpha \in L$ is done by solving a linear system $LX = \alpha$ and verifying that X has integer entries.

Inclusion (Algorithm 73): Checking whether a lattice L_1 is included in a lattice L_2 can be done either by checking containment of all 4 basis vectors of L_1 in L_2, or by testing equality of $L_1 + L_2$ and L_2.

Index (Algorithm 74): When $L_1 \subset L_2$, the *index* of L_1 in L_2, denoted by $[L_2 : L_1]$, is the order of the finite quotient group L_2/L_1. This value equals $|\det(L_1)/\det(L_2)|$ and can be computed using the determinant algorithm for dimension 4.

Basis reduction (Algorithm 78): We say a lattice is *reduced* when its basis is reduced according to the definition of Section 3.1.3 instantiated with the bilinear form of Eq. (3.2). We use the $\mathrm{L2}_{\eta,\delta}$ algorithm to compute a reduced basis. The Gram matrix of the basis $(1, \mathbf{i}, \mathbf{j}, \mathbf{k})$ is the diagonal matrix G of entries $(2, 2, 2p, 2p)$. If another basis $(\alpha_1, \alpha_2, \alpha_3, \alpha_4)$ is written as $(\alpha_1\ \alpha_2\ \alpha_3\ \alpha_4) = (1\ \mathbf{i}\ \mathbf{j}\ \mathbf{k})M$ for some matrix M, then its Gram matrix is $G' = M^t G M$.

3.2.3 Quaternion orders and ideals

An *order* is a lattice of $\mathcal{B}_{p,\infty}$ that is also a subring (i.e., closed under multiplication). Elements of an order $\mathcal{O}$ are said to be integral, since they have reduced trace and norm in $\mathbb{Z}$. An order is called *maximal* when it is not contained in any other larger order. The quaternion algebra used in QIMEN-PRISM contains a maximal order with basis $(1, \mathbf{i}, \frac{\mathbf{i}+\mathbf{j}}{2}, \frac{1+\mathbf{k}}{2})$ which will be denoted by $\mathcal{O}_0$ in the remainder of this section. $\mathcal{O}_0$ contains a (non-maximal) suborder of basis $(1, \mathbf{i}, \mathbf{j}, \mathbf{k})$.

Let $\mathcal{O}$ be an order. A left (right) *integral ideal* of $\mathcal{O}$, henceforth only called a left (right) ideal of $\mathcal{O}$, is a sublattice of $\mathcal{O}$, closed under multiplication by $\mathcal{O}$ on the left (right). The left order of an ideal is defined as

$$\mathcal{O}_L(I) = \{\alpha \in \mathcal{B}_{p,\infty} \mid \alpha I \subset I\}$$

and similarly for the right order $\mathcal{O}_R(I)$. In this case, I is clearly a left ideal of $\mathcal{O}_L(I)$. An ideal with left order $\mathcal{O}_L$ and right order $\mathcal{O}_R$ is called a *connecting ideal* of $\mathcal{O}_L$ and $\mathcal{O}_R$, written as $(\mathcal{O}_L, \mathcal{O}_R)$-ideal. The *reduced norm* of an ideal I, denoted by $\mathrm{nrd}(I)$, is the greatest common divisor of the reduced norms of its elements; it is an integer equal to

$$\sqrt{[\mathcal{O}_L(I) : I]} = \sqrt{[\mathcal{O}_R(I) : I]}.$$

Any ideal can be written as

$$I = \mathcal{O}_L(I)\alpha + \mathcal{O}_L(I)\ \mathrm{nrd}(I)$$

for some $\alpha \in \mathcal{O}_L(I)$, and similarly for $\mathcal{O}_R(I)$. We simplify this notation by writing $\mathcal{O}\alpha + \mathcal{O}N = \mathcal{O}(\alpha, N)$ for any order $\mathcal{O}$.

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

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

We represent ideals by their lattices and compute equality, membership, sum, intersection and multiplication like for them. We can use these elementary operations to compute the ideal $\mathcal{O}(\alpha, N)$: use lattice multiplication to compute $\mathcal{O}\alpha$ and $\mathcal{O}N$, then lattice sum to compute $\mathcal{O}(\alpha, N)$. Conversely, given an ideal I of reduced norm N, a quaternion $\alpha \in I$ such that $I = \mathcal{O}(\alpha, N)$ can be found by taking arbitrary elements in I until one such that $\gcd(\mathrm{nrd}(\alpha), N^2) = N$ is found. This method is used in the reference implementation and detailed in IdealGenerator.

In what follows we describe some algorithms for ideals that are used in QIMEN-PRISM. Details of these algorithms are given in Section F.2.

Ideal inverse (Algorithm 83): Since all ideals in QIMEN-PRISM are connecting maximal orders, they have inverses. The *inverse* of such an ideal I is

$$I^{-1} = \frac{1}{\mathrm{nrd}(I)}\,\overline{I}$$

where $\overline{I}$ is the lattice of conjugates of elements in I. I^{-1} is not an ideal, but a rank-4 lattice, and for lattice multiplication, $II^{-1} = \mathcal{O}_L(I)$ and $I^{-1}I = \mathcal{O}_R(I)$.

The left and right order of an ideal (Algorithm 84): Since all ideals in QIMEN-PRISM are ideals connecting maximal orders, the left and right orders of an ideal I can be computed using its inverse, i.e., $\mathcal{O}_L(I) = II^{-1}$ and $\mathcal{O}_R(I) = I^{-1}I$.

Computing a connecting ideal (Algorithm 87): Given two orders $\mathcal{O}_L$ and $\mathcal{O}_R$, we compute a connecting ideal as

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

where N is the square root of the index of $\mathcal{O}_L \cap \mathcal{O}_R$ in $\mathcal{O}_L$.

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

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

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

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

It is readily verified that $[J]^*([J]_*I) = I$.

3.2.4 Norm equations

Finding quaternion elements with given norm is an important subroutine in RandFixNormIdeal; we introduce the algorithm GeneralizedRepresentInteger for this task. Throughout this section, we work with a specific quaternion order $\mathcal{O}_0 := \mathbb{Z} + \mathbf{i}\mathbb{Z} + \frac{\mathbf{i}+\mathbf{j}}{2}\mathbb{Z} + \frac{1+\mathbf{k}}{2}\mathbb{Z}$, where p is a prime such that $p \equiv 3 \bmod 4$.

3.2.4.1 Cornacchia's algorithm

Cornacchia's algorithm [Cor08] allows us to efficiently find integer solutions for equations of the form $x^2 + qy^2 = m$ with q, m positive integers, provided that enough factorization information on m is available. In the routines below, we only need the case $q = 1$, i.e., representations of integers as sums of two squares. For prime m, an algorithm following [MN90] is given in Cornacchia. We also use a composite-input wrapper, CornacchiaGeneral, which first removes powers of a fixed precomputed list of small primes and then applies Cornacchia to the remaining cofactor when possible.

Algorithm 7 Cornacchia(m)

Input: A prime integer m
Output: $(x, y) \in \mathbb{Z}^2$ such that $x^2 + y^2 = m$, or $\perp$ if no solution is found

if $m = 2$ **then**
  return $(1, 1)$
if $m \not\equiv 1 \pmod 4$ **then** ▷ -1 is not a square modulo m
  return $\perp$
$r \leftarrow$ ModularSQRT$(-1, m)$
$s \leftarrow m$
while $r^2 \geq m$ **do**
  $(r, s) \leftarrow (s \bmod r, r)$
$x \leftarrow r$
$Y \leftarrow m - r^2$
if Y is not a square in $\mathbb{Z}$ **then**
  return $\perp$
$y \leftarrow \sqrt{Y}$
return (x, y)

The algorithm below uses a precomputed list

$$\mathcal{P} = (p_0, \ldots, p_{s-1})$$

of small primes. For QIMEN-PRISM, this list consists of 2 together with the first 100 odd primes $\ell \equiv 1 \pmod 4$, for a total of 101 entries. The list is fixed by the parameter set and we denote it by `cornacchia_prime_list`, i.e., $\mathcal{P}$ is taken to be `cornacchia_prime_list` in QIMEN-PRISM. Inside CornacchiaGeneral, the variable z is a Gaussian integer $z = u+v\sqrt{-1} \in \mathbb{Z}[\sqrt{-1}]$; hence $\mathrm{Re}(z) = u$ and $\mathrm{Im}(z) = v$ denote its two integer coefficients.

Algorithm 8 CornacchiaGeneral$(m, \mathcal{P})$

Input: A positive integer m and the precomputed list $\mathcal{P} = (p_0, \ldots, p_{s-1})$
Output: $(x, y) \in \mathbb{Z}^2$ such that $x^2 + y^2 = m$, or $\perp$ if no solution is found

$m' \leftarrow m$ and $e_i \leftarrow 0$ for each $p_i \in \mathcal{P}$
for i from 0 up to $s-1$ **do**
    while $p_i \mid m'$ **do**
        $m' \leftarrow m'/p_i$
        $e_i \leftarrow e_i + 1$
if $m' = 1$ **then**
    $z \leftarrow 1$
else
    if PrimalityTest$(m') =$ False or $m' \not\equiv 1 \pmod 4$ **then**
        return $\perp$
    $(x, y) \leftarrow$ Cornacchia(m')
    if $(x, y) = \perp$ **then**
        return $\perp$
    $z \leftarrow x + y\sqrt{-1}$
for i from 0 up to $s-1$ **do**
    if $e_i > 0$ **then**
        $(a_i, b_i) \leftarrow$ Cornacchia(p_i)
        if $(a_i, b_i) = \perp$ **then**
            return $\perp$
        $z \leftarrow z(a_i + b_i\sqrt{-1})^{e_i}$
return $(\mathrm{Re}(z), \mathrm{Im}(z))$

3.2.4.2 Representing integers by a special extremal order

Our goal is to find $x, y, z, t \in \mathbb{Z}$ such that the norm of the quaternion element $\gamma := x + y\mathbf{i} + z\frac{\mathbf{i}+\mathbf{j}}{2} + t\frac{1+\mathbf{k}}{2}$ equals a fixed integer M, often taken to be larger than p.

One observation is that, to solve the equation $\mathrm{nrd}(\gamma) = (x+t/2)^2+(y+z/2)^2+p((t/2)^2+$

$(z/2)^2) = M$, it is equivalent to solving

$$x^2 + y^2 + p(z^2 + t^2) = 4M \tag{3.3}$$

with an integer solution (x, y, z, t) such that $x \equiv t \bmod 2$ and $y \equiv z \bmod 2$. Then, this element $\gamma = \frac{x-t}{2} + \frac{y-z}{2}\mathbf{i} + z\frac{\mathbf{i}+\mathbf{j}}{2} + t\frac{1+\mathbf{k}}{2} = \frac{x+y\mathbf{i}+z\mathbf{j}+t\mathbf{k}}{2}$.

Hence, it suffices to find a suitable solution for Eq. (3.3). The algorithm GeneralizedRepresentInteger proceeds as follows: it first samples z, t in an appropriate range as specified in the algorithm, then computes the value $M' := 4M - p(z^2 + t^2)$. Once M' is a prime congruent to 1 modulo 4, it calls Cornacchia to solve the binary quadratic equation $x^2 + y^2 = M'$.

Algorithm 9 GeneralizedRepresentInteger(M)

Input: $M \in \mathbb{Z}$ odd such that $M > p$
Output: $\gamma \in \mathcal{O}_0$ with $\mathrm{nrd}(\gamma)$ equal to M, or raises an exception

Counter $\leftarrow 0$, bound $\leftarrow \left\lceil \frac{4M}{p} \right\rceil$, and `found` $\leftarrow$ `false`
while (`found` $=$ `false`) and (`counter` $<$ `bound`) **do**
    `counter` $\leftarrow$ `counter` $+ 1$
    Sample z uniformly from $\left[1, \ldots, \left\lfloor \sqrt{\frac{4M}{p}} \right\rfloor\right]$
    Sample t uniformly from $[-m', \ldots, m']$ for $m' = \left\lfloor \sqrt{\frac{4M - pz^2}{p}} \right\rfloor$
    Set $M' \leftarrow 4M - p(z^2 + t^2)$
    if PrimalityTest(M') **then**
        if $M' \equiv 1 \pmod 4$ **then**
            `found` $\leftarrow$ `true`
            $(x, y) \leftarrow$ Cornacchia(M')
            if $x \neq t \bmod 2$ or $y \neq z \bmod 2$ **then**
                $(x, y) \leftarrow (y, x)$
            $\gamma \leftarrow \frac{x+y\mathbf{i}+z\mathbf{j}+t\mathbf{k}}{2}$
if `found` **then**
    return γ
else
    raise Exception("`GeneralizedRepresentInteger failed`")

Remark 3.2.1. *Since $x^2 + y^2 = M' \equiv 1 \pmod 4$, the integers x and y have opposite parity. Moreover, from $M' = 4M - p(z^2 + t^2)$, where p and M' are odd, the integers z and t also have opposite parity. Hence, either $x \equiv t \pmod 2$ and $y \equiv z \pmod 2$, or the two assignments are exchanged. In the latter case, replacing (x, y) with (y, x) gives the required congruences. The swap preserves the Cornacchia equation, since $x^2 + y^2 = y^2 + x^2$. Therefore, after the possible swap, $\gamma = (x + y\mathbf{i} + z\mathbf{j} + t\mathbf{k})/2$ belongs to $\mathcal{O}_0$.*

A similar idea is used in [AAA+25], but only under an architecture-specific condition, and it is not used in [BBC+26]. By incorporating this additional swap mechanism, QIMEN-PRISM achieves a 15%–20% reduction in signing time across the security levels.

3.2.4.3 Computing ideals

QIMEN-PRISM requires to compute two different kinds of random ideals taken care by the following two algorithms:

RandFixNormIdeal Given an integer N that is sufficiently large and not a multiple of p, this algorithm samples uniformly at random a left $\mathcal{O}_0$-ideal of reduced norm N.

In the case when N is not a prime, should be called with the parameter `prime` set to `false` and RandFixNormIdeal first uses GeneralizedRepresentInteger to find an element γ of $\mathcal{O}_0$ of norm N. If GeneralizedRepresentInteger fails, the algorithm is intended to propagate the failure. It then multiplies this quaternion with a uniformly random quaternion β in $\mathcal{O}_0/N\mathcal{O}_0$ of norm coprime to N. The result $\gamma\beta$ still has norm divisible by N. The left $\mathcal{O}_0$-ideal generated by N and $\gamma\beta$ is then returned by the algorithm. In order to use GeneralizedRepresentInteger, N should be big enough relative to p. This condition is satisfied in the use cases of QIMEN-PRISM.

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

Remark 3.2.2. *In QIMEN-PRISM, RandFixNormIdeal is called with `prime` set to be `false` only in QIMEN-PRISM.GenIso with $N = q(a^a - q)$. Even though not specified in the paper, we observe that in* salt*-PRISM [BBC+26], in this case, the input to GeneralizedRepresentInteger in Line 10 is $q^2(2^a - q)$ instead of $q(2^a - q)$. We change this to $q(2^a - q)$ in QIMEN-PRISM. This does not affect correctness, nor does it weaken security: the subsequent step samples β, which randomizes the returned ideal. We observe in our experiments that this modification makes QIMEN-PRISM.Sign around 12% to 17% times faster for different security levels for NGCC. This is due to the fact that a smaller input to GeneralizedRepresentInteger makes the probability of finding a prime higher and also speeds up the primality test step.*

Algorithm 10 RandFixNormIdeal(N, `prime`)

Input: A positive integer N not divisible by p which is the norm of some left $\mathcal{O}_0$ ideal and a boolean `prime` indicating whether N is prime.

Output: A random left ideal J' of $\mathcal{O}_0$ of norm N, or raise an exception.

`found` $\leftarrow$ `false`
if `prime` **then**
  while not `found` **do**
    $g_1, g_2, g_3 \leftarrow$ independent uniformly random integers in $[0, N-1]$
    $\gamma \leftarrow g_1\mathbf{i} + g_2\mathbf{j} + g_3\mathbf{k}$
    `found` $\leftarrow (1 = \mathsf{Legendre}(-\mathrm{nrd}(\gamma), N))$
    if `found` **then**
      $\gamma \leftarrow \gamma +$ ModularSQRT$(-\mathrm{nrd}(\gamma), N)$
else
  $\gamma \leftarrow$ GeneralizedRepresentInteger(N)
while not `found` **do**
  $x, y, z, w \leftarrow$ uniformly randomly selected integers in $[1, N]$
  $\beta \leftarrow x + y\mathbf{i} + z\mathbf{j} + w\mathbf{k}$
  `found` $\leftarrow (\gcd(\mathrm{nrd}(\beta), N) = 1)$
$J' \leftarrow$ the left $\mathcal{O}_0$-ideal generated by $\gamma\beta$ and N
return J'

RandomEquivalentPrimeIdeal Given a left $\mathcal{O}_0$-ideal, this algorithm finds an equivalent left $\mathcal{O}_0$-ideal J such that $\mathrm{nrd}(J)$ is a small prime. Note that in this algorithm, the distribution of the output J among all equivalent ideals does not matter for the security of the scheme. Given an integral ideal I, RandomEquivalentPrimeIdeal finds an equivalent ideal J (i.e., $I = J\alpha$, where $\alpha \in \mathcal{B}^{\times}_{p,\infty}$) with different (i.e., prime and bounded) norm. To do so, it employs the surjection

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

from $I \setminus \{0\}$ to the set of ideals J equivalent to I. The algorithm samples constants c_i within the bound $[-$`QUAT_repres_bound_input`$,$ `QUAT_repres_bound_input`$]$ and constructs $\beta = \sum_{i=1}^{4} c_i\alpha_i$, where $(\alpha_1, \ldots, \alpha_4)$ is a reduced basis of I, before outputting $J = \chi_I(\beta)$ if the norm of J is prime. Heuristically, we can expect the norm of the output ideal to be $\approx \sqrt{p}$ which should be much smaller than the norm of the input in QIMEN-PRISM. If no prime-norm equivalent ideal is found within the prescribed number of trials (determined by the algorithm parameter `QUAT_repres_bound_input` specified in Section 4.1.1), the algorithm reports failure.

Algorithm 11 RandomEquivalentPrimeIdeal(I)

Input: I, a left $\mathcal{O}_0$-ideal.
Output: $J \sim I$ of small prime norm, or raise an exception if unsuccessful.

Initialize `counter` $\leftarrow 0$
$G_I \leftarrow \text{Gram}(I)$.
$((\alpha_1, \alpha_2, \alpha_3, \alpha_4), \_) \leftarrow \text{L2}_{\eta,\delta}(I, G_I)$
while `counter` $< (2 \cdot$ `QUAT_equiv_bound_coeff` $+ 1)^4$ **do**
    `counter` $\leftarrow$ `counter` $+ 1$
    Sample c_1, c_2, c_3, c_4 uniformly at random from $[-b, \ldots, b]$, with bound $b =$ `QUAT_equiv_bound_coeff`
    $\beta \leftarrow \sum_{i=1}^{4} c_i \alpha_i$
    $J \leftarrow \chi_I(\beta)$
    if $\text{nrd}(J)$ is prime **then return** J
raise Exception: (`"RandomEquivalentPrimeIdeal failed"`)

3.3 Ideal-to-isogeny translation

In this section, we explain how to translate quaternion ideals into isogenies. We first recall the correspondence between ideals and isogenies in Section 3.3.1. We then describe a representation of isogenies in dimension 2 in Section 3.3.2 and the Qlapoti algorithm, a key subroutine for the translation, in Section 3.3.3. Finally, in Section 3.3.4, we present IdealToIso, an efficient algorithm that translates an ideal into its corresponding isogeny.

3.3.1 Correspondence of ideals and isogenies

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

Let E be an elliptic curve and let $\mathcal{O}$ be an order equipped with an isomorphism $\mathcal{O} \simeq \text{End}(E)$. This data gives a bijection between

$$\left\{ \begin{array}{c} \text{left ideals of } \mathcal{O} \\ \text{of norm coprime to } p \end{array} \right\} \quad \text{and} \quad \left\{ \begin{array}{c} \text{finite subgroups} \\ \text{of } E \end{array} \right\}.$$

Combining this with the correspondence between finite subgroups and separable isogenies gives a bijection

$$\left\{ \begin{array}{c} \text{left ideals of } \mathcal{O} \\ \text{of norm coprime to } p \end{array} \right\} \quad \longleftrightarrow \quad \left\{ \begin{array}{c} \text{separable isogenies} \\ \text{from } E \end{array} \right\} / \sim,$$

where $\sim$ identifies isogenies that differ by post-composition with an isomorphism. Explicitly, an ideal I is sent to the finite subgroup

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

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

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

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

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

In QIMEN-PRISM, we use a prime $p \equiv 3 \pmod 4$. For such a prime, the curve

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

is supersingular, and its endomorphism ring $\mathrm{End}(E_0)$ is isomorphic to

$$\mathcal{O}_0 := \mathbb{Z} \oplus \mathbf{i}\mathbb{Z} \oplus \frac{\mathbf{i}+\mathbf{j}}{2}\mathbb{Z} \oplus \frac{1+\mathbf{k}}{2}\mathbb{Z}.$$

One such isomorphism can be given by sending $\mathbf{j}$ to the Frobenius endomorphism $(x, y) \mapsto (x^p, y^p)$, and $\mathbf{i}$ to the automorphism $(x, y) \mapsto (-x, \sqrt{-1}y)$ on E_0, where $\sqrt{-1}$ is a square root of -1 in $\mathbb{F}_{p^2}$. For the remainder of this document, we fix this choice of E_0, $\mathcal{O}_0$, and the isomorphism $\mathrm{End}(E_0) \simeq \mathcal{O}_0$. The symbol $\mathcal{O}$ denotes an arbitrary maximal order, possibly equal to $\mathcal{O}_0$. Suppose we are given an isogeny $\varphi : E_0 \to E$ and the corresponding $(\mathcal{O}_0, \mathcal{O})$-ideal I. Then the fixed isomorphism $\mathcal{O}_0 \simeq \mathrm{End}(E_0)$ induces an isomorphism $\mathcal{O} \simeq \mathrm{End}(E)$. Let $N = \mathrm{nrd}(I)$. Since I is an $(\mathcal{O}_0, \mathcal{O})$-ideal, we have $N\mathcal{O} \subseteq \mathcal{O}_0$. Thus, this induced isomorphism is obtained by first viewing $N\alpha \in N\mathcal{O}$ as an element of $\mathcal{O}_0$, and then sending it to

$$\varphi \circ \alpha \circ \widehat{\varphi} \in \mathrm{End}(E).$$

The isomorphism on all of $\mathcal{O}$ is then obtained by extending scalars. In the following algorithms, we either work with E_0 or with a curve E for which such a pair (φ, I) is known. Thus, we do not explicitly specify the isomorphism used in the correspondence; it is always the implicitly defined isomorphism $\mathcal{O} \simeq \mathrm{End}(E)$.

3.3.2 Representation of isogenies in dimension 2

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

$$\begin{array}{ccc} E_1 & \xrightarrow{\varphi_1} & F_1 \\ {\scriptstyle\varphi_2}\downarrow & & \downarrow{\scriptstyle\varphi_2'} \\ F_2 & \xrightarrow[\varphi_1']{} & E_2 \end{array} \tag{3.4}$$

where $\deg(\varphi_1) = \deg(\varphi_1') = d_1$ and $\deg(\varphi_2) = \deg(\varphi_2') = d_2$. Then the isogeny

$$\Phi : E_1 \times E_2 \to F_1 \times F_2$$

defined by $\begin{pmatrix} \varphi_1 & \widehat{\varphi_2'} \\ -\varphi_2 & \widehat{\varphi_1'} \end{pmatrix}$ is a $(2^a, 2^a)$-isogeny with kernel

$$\{([d_1]P, \varphi_2' \circ \varphi_1(P)) \mid P \in E_1[2^a]\}. \tag{3.5}$$

Given E_1, E_2, d_1, d_2, e, and the restriction of ψ on $E_1[2^{a+2}]$, we can compute an isogeny $\iota \circ \Phi$ by Isogeny22Chain for an isomorphism ι between products of elliptic curves.

In the verification of QIMEN-PRISM, it is needed that given $(E_1, P_1, Q_1, E_2, P_2, Q_2, a, q)$, the verifier checks whether there exists an isogeny $\varphi : E_1 \to E_2$ of degree $q(2^a - q)$ such that $\varphi(P_1) = [q]P_2$ and $\varphi(Q_1) = [q]Q_2$. If this were true, then this isogeny φ would be naturally seen as being derived from a commutative diagram like Eq. (3.4) with $d_1 = q$ (or $2^a - q$) and $d_2 = 2^a - q$ (or q). Therefore, the existence of φ can be verified by computing $\iota \circ \Phi$, and if it exists, then checking whether the first component of $\iota \circ \Phi(P, 0_{E_2})$ is $\varphi(P)$ for an isogeny φ of degree q or $2^a - q$ and $P \in E_1[2^a]$. In the end, one can determine whether this is the case by computing the Weil pairing as described in the next paragraph.

Let (P, Q) be a basis of $E_1[2^a]$ and $(P_1, P_2) = \iota \circ \Phi(P, 0_{E_2})$, $(Q_1, Q_2) = \iota \circ \Phi(Q, 0_{E_2})$. Then, we have

$$e_{2^a}(P_1, Q_1) = e_{2^a}(P, Q)^{d_1}.$$

Since $2^a > q, 2^a - q$ and $e_{2^a}(P, Q)$ is the primitive 2^a-th root of unity, the value of d_1 can be determined by checking whether $e_{2^a}(P_1, Q_1)$ is equal to $e_{2^a}(P, Q)^q$ or $e_{2^a}(P, Q)^{2^a - q}$. If it is equal to one of them, then the existence of φ is verified; otherwise, it is not. In fact, this is also how the verification is done in CheckIsogeny. If this check fails, the verification algorithm rejects the signature as invalid.

3.3.3 Qlapoti algorithm

The Qlapoti algorithm, introduced in [BCRSE+26], serves as a key subroutine in IdealToIso. Given a left $\mathcal{O}_0$-ideal J and a positive integer k, it computes two left $\mathcal{O}_0$-ideals I_1 and I_2, both equivalent to J, such that

$$\mathrm{nrd}(I_1) + \mathrm{nrd}(I_2) = 2^k.$$

Algorithm 12 CheckIsogeny$(E_1, P_1, Q_1, E_2, P_2, Q_2, a, q)$

Input: $(E_1, P_1, Q_1, E_2, P_2, Q_2, a, q)$ such that (P_1, Q_1) is a basis of $E_1[2^{a+2}]$ and (P_2, Q_2) is a basis of $E_2[2^{a+2}]$.

Output: true if there exists an isogeny $\varphi : E_1 \to E_2$ of degree $q(2^a - q)$ such that $\varphi(P_1) = [q]P_2$ and $\varphi(Q_1) = [q]Q_2$, and false otherwise.

$K_1 \leftarrow ([q]P_1, P_2)$
$K_2 \leftarrow ([q]Q_1, Q_2)$
try
    $\_, [(P', \_), (Q', \_)] \leftarrow$ Isogeny22Chain$(K_1, K_2, [(P_1, 0_{E_2}), (Q_1, 0_{E_2})])$
catch
    return false
if $e_{2^a}(P', Q') = e_{2^a}(P_1, Q_1)^m$ for $m = q$ or $2^a - q$ **then**
    return true
return false

In QIMEN-PRISM, we use this algorithm with $k = e - 2$, where e is the parameter defined in Section 4.1.1. Equivalently, the algorithm seeks $\beta_1, \beta_2 \in J$ satisfying

$$\mathrm{nrd}(\beta_1) + \mathrm{nrd}(\beta_2) = 2^k \,\mathrm{nrd}(J). \tag{3.6}$$

We now sketch the Qlapoti algorithm for finding β_1, β_2.

Write $J = \mathcal{O}_0\langle N, \alpha\rangle$, where $N = \mathrm{nrd}(J)$. Then, for $i = 1, 2$, one can write

$$\beta_i = \gamma_i N + \gamma_i' \alpha$$

with $\gamma_i, \gamma_i' \in \mathcal{O}_0$. Restricting to $\gamma_i = a_i + b_i\mathbf{i}$ and $\gamma_i' = 1$, and writing $\alpha = a_\alpha + b_\alpha\mathbf{i} + c_\alpha\mathbf{j} + d_\alpha\mathbf{k}$, Eq. (3.6) is equivalent to the following equation:

$$N \cdot (a_1^2 + b_1^2 + a_2^2 + b_2^2) + 2a_\alpha(a_1 + a_2) + 2b_\alpha(b_1 + b_2) = 2^k - 2r, \tag{3.7}$$

where $r = \mathrm{nrd}(\alpha)/N$.

The first step of solving Eq. (3.7) is to find a small solution (s, t) to

$$2a_\alpha x + 2b_\alpha y = 2^k - 2r \pmod N.$$

We refer to this as the short-congruence step, and we expand this in QlapotiShort-Congruence. The QlapotiShortCongruence algorithm also uses a two-dimensional Gauss reduction procedure Dim2ReducedBasis of [Coh93, Algorithm 3.1.14], where an ordered basis $(\boldsymbol{b}_0, \boldsymbol{b}_1)$ of a rank-two lattice in $\mathbb{Z}^2$ is *reduced* if

$$\|\boldsymbol{b}_0\|^2 \le \|\boldsymbol{b}_1\|^2 \qquad \text{and} \qquad |\boldsymbol{b}_0 \cdot \boldsymbol{b}_1| \le \frac{1}{2}\|\boldsymbol{b}_0\|^2.$$

Equivalently, $\boldsymbol{b}_1$ has been size-reduced with respect to $\boldsymbol{b}_0$ by nearest-integer rounding, and $\boldsymbol{b}_0$ is no longer than $\boldsymbol{b}_1$.

Algorithm 13 Dim2ReducedBasis(L)

Input: A rank-two lattice $L = (\boldsymbol{a}, \boldsymbol{b}) \subseteq \mathbb{Z}^2$.
Output: A reduced basis of L.

if $\|\boldsymbol{a}\|^2 < \|\boldsymbol{b}\|^2$ **then**
    Swap $\boldsymbol{a}$ and $\boldsymbol{b}$
while true **do**
    $r \leftarrow \mathrm{round}((\boldsymbol{a} \cdot \boldsymbol{b})/\|\boldsymbol{b}\|^2)$ ▷ nearest-integer rounding
    $\boldsymbol{t} \leftarrow \boldsymbol{a} - r\boldsymbol{b}$
    if $\|\boldsymbol{t}\|^2 < \|\boldsymbol{b}\|^2$ **then**
        $\boldsymbol{a} \leftarrow \boldsymbol{b}$ and $\boldsymbol{b} \leftarrow \boldsymbol{t}$
    else
        break
if $\|\boldsymbol{t}\|^2 < \|\boldsymbol{a}\|^2$ **then**
    $\boldsymbol{a} \leftarrow \boldsymbol{t}$
return $(\boldsymbol{b}, \boldsymbol{a})$

Algorithm 14 QlapotiShortCongruence(A, B, M, N)

Input: Integers A, B, M, N such that at least one of A and B is invertible modulo N.
Output: A short pair $(s, t) \in \mathbb{Z}^2$ such that $As + Bt \equiv M \pmod{N}$.

swapped $\leftarrow$ false
if $\gcd(A, N) \neq 1$ **then**
    Swap A and B; swapped $\leftarrow$ true
$u \leftarrow A^{-1} \bmod N$
$x \leftarrow Bu \bmod N$ and $T \leftarrow Mu \bmod N$
$L \leftarrow \langle (N - x, 1), (N, 0) \rangle \subseteq \mathbb{Z}^2$ ▷ a rank-two integer lattice
$R \leftarrow$ Dim2ReducedBasis(L)
$D \leftarrow \det(R)$ and $R_{\mathrm{inv}} \leftarrow D \cdot R^{-1}$ ▷ integer numerator of R^{-1}
$\boldsymbol{v} \leftarrow (-T, 0)$
$(c_1, c_2) \leftarrow R_{\mathrm{inv}}\boldsymbol{v}$
$c_i \leftarrow \mathrm{round}(c_i/D)$ for $i = 1, 2$ ▷ nearest-integer rounding
$\boldsymbol{c} \leftarrow (c_1, c_2)$
$\boldsymbol{w} \leftarrow R\boldsymbol{c}$
$(s, t) \leftarrow \boldsymbol{w} - \boldsymbol{v}$
if swapped **then**
    Swap s and t
return (s, t)

Having found s and t, substituting $a_2 = s - a_1$ and $b_2 = t - b_1$, dividing by N, and multiplying by 2, Eq. (3.7) then becomes a standard sum-of-squares problem

$$(2a_1 - s)^2 + (2b_1 - t)^2 = \frac{2(2^k - 2r - 2a_\alpha s - 2b_\alpha t)}{N} - s^2 - t^2,$$

which can be solved using Cornacchias algorithm. Solving for a_1, b_1, and then computing a_2, b_2 gives β_1, β_2. This algorithm is detailed in Qlapoti. This algorithm does not always succeed, and we provide a detailed failure analysis in Chapter 9.

Algorithm 15 QLAPOTI(J, k)

Input: A left $\mathcal{O}_0$-ideal J, an integer k.
Output: Elements $\beta_1, \beta_2 \in J$ satisfying
$\operatorname{nrd}(\beta_1) + \operatorname{nrd}(\beta_2) = 2^k \operatorname{nrd}(J)$ and $\gcd(\operatorname{nrd}(\beta_1)/\operatorname{nrd}(J), \operatorname{nrd}(\beta_2)/\operatorname{nrd}(J)) = 1$.
or raises an exception.

$I \leftarrow$ RANDOMEQUIVALENTPRIMEIDEAL(J)
$N_I \leftarrow \operatorname{nrd}(I)$
`counter` $\leftarrow 0$, `bound` $\leftarrow \left\lceil 0.607927 N_I \cdot \frac{2^k}{4p} \right\rceil$
while `counter` $<$ `bound` **do**
  `counter` $\leftarrow$ `counter` $+ 1$
  $\alpha \leftarrow$ IDEALGENERATOR(I)
  $a_\alpha, b_\alpha, c_\alpha, d_\alpha \leftarrow$ the coordinates of α in $1, \mathbf{i}, \mathbf{j}, \mathbf{k}$.
  $r \leftarrow \operatorname{nrd}(\alpha)/N_I$
  if $\gcd(2a_\alpha, N_I) \neq 1$ and $\gcd(2b_\alpha, N_I) \neq 1$ **then**
    continue
  $(s, t) \leftarrow$ QLAPOTISHORTCONGRUENCE$(2a_\alpha, 2b_\alpha, 2^k - 2r, N_I)$
  $z \leftarrow 2(2^k - 2r - 2a_\alpha s - 2b_\alpha t)/N_I - s^2 - t^2$
  if $z < 0$ **then**
    continue
  if $z \equiv 0 \pmod 4$ and not $s = t = 0 \pmod 2$ **then**
    continue
  if $z \equiv 1 \pmod 4$ and not $s \neq t \pmod 2$ **then**
    continue
  if $z \equiv 2 \pmod 4$ and not $s = t = 1 \pmod 2$ **then**
    continue
  if $z \equiv 3 \pmod 4$ **then**
    continue
  `sol` $\leftarrow$ CORNACCHIAGENERAL(z)
  if `sol` $= \perp$ **then**
    continue
  $z_0, z_1 \leftarrow$ `sol`
  if $z_0 \not\equiv s \pmod 2$ **then**
    swap z_0, z_1
  $a_1 \leftarrow (z_0 + s)/2, \quad b_1 \leftarrow (z_1 + t)/2$
  $a_2 \leftarrow s - a_1, \quad b_2 \leftarrow t - b_1$
  $\beta_1 \leftarrow N_I(a_1 + b_1\mathbf{i}) + \alpha, \quad \beta_2 \leftarrow N_I(a_2 + b_2\mathbf{i}) + \alpha$
  return β_1, β_2
raise Exception(`"Qlapoti failed"`)

Remark 3.3.1 (Alternative representatives and implementation failure)**.** *A Cornacchia*

solution $z = z_0^2 + z_1^2$ determines further solutions by independently changing the signs of z_0 and z_1. Under the back-substitution above, these sign changes exchange a_1 with a_2 and/or b_1 with b_2. Hence, the same Cornacchia solution gives the four ordered representations

$$(a_1, b_1; a_2, b_2), \qquad (a_1, b_2; a_2, b_1), \qquad (a_2, b_1; a_1, b_2), \qquad (a_2, b_2; a_1, b_1).$$

The corresponding pairs of quaternion elements are

$$\begin{aligned}
(\beta_1^{(0)}, \beta_2^{(0)}) &= (N_I(a_1 + b_1\mathbf{i}) + \alpha,\ N_I(a_2 + b_2\mathbf{i}) + \alpha),\\
(\beta_1^{(1)}, \beta_2^{(1)}) &= (N_I(a_1 + b_2\mathbf{i}) + \alpha,\ N_I(a_2 + b_1\mathbf{i}) + \alpha),\\
(\beta_1^{(2)}, \beta_2^{(2)}) &= (N_I(a_2 + b_1\mathbf{i}) + \alpha,\ N_I(a_1 + b_2\mathbf{i}) + \alpha),\\
(\beta_1^{(3)}, \beta_2^{(3)}) &= (N_I(a_2 + b_2\mathbf{i}) + \alpha,\ N_I(a_1 + b_1\mathbf{i}) + \alpha).
\end{aligned}$$

All four pairs satisfy the norm equation in the output condition. The subsequent implementations have an additional condition to the pair $(\beta_1^{(v)}, \beta_2^{(v)}), v \in \{0, 1, 2, 3\}$. Thus, a pair that is a valid output of the abstract Qlapoti algorithm may still be rejected by the subsequent subroutines. QIMEN-PRISM exploits multiple solutions in the Cornacchia subroutine to reduce the loop, which accelerates the keygen and the signing by 5% to 9% up to the security level.

3.3.4 Ideal-to-isogeny algorithm

IdealToIso is the main algorithm of this section. Given a left $\mathcal{O}_0$-ideal J, it computes the codomain E_J of the corresponding isogeny φ_J, together with the evaluation of φ_J on a fixed torsion basis specified below.

Precomputed data. The algorithm assumes that $p = f \cdot 2^e - 1$ for a small integer f, and takes a positive integer a such that $a \leq e$. Recall from Section 3.3.1 that $\left(1, \mathbf{i}, \frac{\mathbf{i}+\mathbf{j}}{2}, \frac{1+\mathbf{k}}{2}\right)$ is the fixed integral basis of $\mathcal{O}_0$. When IdealToIso is used in QIMEN-PRISM, the following data are precomputed:

- a basis (P_0, Q_0) of $E_0[2^{a+2}]$, together with matrices $M_1, \ldots, M_4 \in \mathsf{M}_2(\mathbb{Z}/2^{a+2}\mathbb{Z})$ representing the actions of these four basis elements with respect to (P_0, Q_0);
- a basis (P_0^e, Q_0^e) of $E_0[2^e]$, together with matrices $M_1^e, \ldots, M_4^e \in \mathsf{M}_2(\mathbb{Z}/2^e\mathbb{Z})$ representing the same actions with respect to (P_0^e, Q_0^e).

Algorithm overview. The algorithm first finds ideals $I_1, I_2 \sim J$ of coprime reduced norms d_1, d_2 satisfying $d_1 + d_2 = 2^{e-2}$ using Qlapoti. Write $I_i = J\frac{\overline{\beta_i}}{\mathrm{nrd}(J)}$ for $i = 1, 2$ and $\beta_i \in J$, then this means that $\widehat{\varphi_{I_i}} \circ \varphi_J = \beta_i$ for $i = 1, 2$. Therefore, the endomorphism $\widehat{\varphi_{I_2}} \circ \varphi_{I_1} = \frac{\beta_2\overline{\beta_1}}{\mathrm{nrd}(J)} \in \mathcal{O}_0$. These isogenies give rise to the following commutative diagram of isogenies between elliptic curves:

Algorithm 16 IdealToIso(J)

Input: Left $\mathcal{O}_0$-ideal J.
Output: Codomain E_J of the isogeny φ_J, and $\varphi_J(P_0), \varphi_J(Q_0)$.

$\beta_1, \beta_2 \leftarrow$ Qlapoti$(J, e-2)$
$d_1 \leftarrow \mathrm{nrd}(\beta_1)/\mathrm{nrd}(J)$
$c_1, c_2, c_3, c_4 \leftarrow$ the coordinates of $\frac{\beta_2\overline{\beta_1}}{\mathrm{nrd}(J)}$ in $1, \mathbf{i}, \frac{\mathbf{i}+\mathbf{j}}{2}, \frac{1+\mathbf{k}}{2}$
$M_\theta \leftarrow c_1 M_1^e + c_2 M_2^e + c_3 M_3^e + c_4 M_4^e$ ▷ $\theta = \frac{\beta_2\overline{\beta_1}}{\mathrm{nrd}(J)}$
$(\widetilde{P_0^e}, \widetilde{Q_0^e})^T \leftarrow M_\theta (P_0^e, Q_0^e)^T$
$K_P \leftarrow ([d_1]P_0^e, \widetilde{P_0^e})$
$K_Q \leftarrow ([d_1]Q_0^e, \widetilde{Q_0^e})$
$E_1 \times E_2,\ [(P, P'), (Q, Q')] \leftarrow$ Isogeny22Chain$(K_P, K_Q, [(P_0, 0_{E_0}), (Q_0, 0_{E_0})])$
if $e_{2^{a+2}}(P, Q) = e_{2^{a+2}}(P_0, Q_0)^{d_1}$ **then**
    $E_J \leftarrow E_1, P_J \leftarrow P, Q_J \leftarrow Q$
else
    $E_J \leftarrow E_2, P_J \leftarrow P', Q_J \leftarrow Q'$
$c_1, c_2, c_3, c_4 \leftarrow$ the coordinates of β_1 in $1, \mathbf{i}, \frac{\mathbf{i}+\mathbf{j}}{2}, \frac{1+\mathbf{k}}{2}$
$M_{\beta_1} \leftarrow c_1 M_1 + c_2 M_2 + c_3 M_3 + c_4 M_4$
$(P_J, Q_J)^T \leftarrow [d_1^{-1} \bmod 2^{a+2}] M_{\beta_1} (P_J, Q_J)^T$
return E_J, P_J, Q_J

$$\begin{array}{ccc} E_0 & \xrightarrow{\varphi_{I_1}} & E_J \\ {\scriptstyle [\varphi_{I_1}]^*\widehat{\varphi_{I_2}}}\downarrow & & \downarrow{\scriptstyle \widehat{\varphi_{I_2}}} \\ E' & \xrightarrow[[[\varphi_{I_1}]^*\widehat{\varphi_{I_2}}]_*\varphi_{I_1}]{} & E_0. \end{array}$$

The isogeny

$$\Phi : E_0 \times E_0 \to E_J \times E'$$

defined by $\begin{pmatrix} \varphi_{I_1} & \widehat{\varphi_{I_2}} \\ -[[\varphi_{I_1}]^*\widehat{\varphi_{I_2}}]_*\varphi_{I_1} & [\widehat{\varphi_{I_2}}]_*\varphi_{I_1} \end{pmatrix}$ is a $(2^{e-2}, 2^{e-2})$-isogeny with kernel

$$\{([d_1]P, \widehat{\varphi_{I_2}} \circ \varphi_{I_1}(P)) \mid P \in E_0[2^{e-2}]\}.$$

To recover the evaluation of the isogeny φ_J on P_0, Q_0, the algorithm first computes Φ, then recovers the evaluation of φ_{I_1} on P_0, Q_0. Finally, since $\widehat{\varphi_{I_i}} \circ \varphi_J = \beta_1$, then $\varphi_J(P_0, Q_0) = [d_1^{-1} \bmod 2^{a+2}]\varphi_{I_1} \circ \beta_i(P_0, Q_0)$.

CHAPTER 4

Protocol

4.1 Protocol Parameters

In this section, we describe QIMEN-PRISM, following the description in [BBC+26, §6].

4.1.1 Parameter requirements

We now introduce some parameters we will use in the rest of document. We will work over a finite field $\mathbb{F}_{p^2}$, where $p = f \cdot 2^e - 1$. We fix the following parameters.

- a is an integer $0 < a < e$. Its exact value is defined in Chapter 5.
- E_0 is the curve with equation $y^2 = x^3 + x$.
- (P_0, Q_0) is a basis of $E_0[2^{a+2}]$.
- N_{sk} is the smallest prime larger than $2^{4\lambda_c}$.
- n_{salt} denotes the bit length of the salt used in Section 4.1.2.
- H_{a-2} is a hash function that maps onto positive integers $< 2^{a-2}$. The concrete hash function used in our construction is described in Chapter 5.
- `FP_ENCODED_BYTES` is the smallest number of bytes capable of representing all elements of $\mathbb{F}_p$; it is defined as $8\lfloor(\log_2(p) + 63)/64\rfloor$.

The auxiliary algorithmic parameters are fixed as follows.

- `QUAT_equiv_bound_coeff` is the coefficient bound used in RandomEquivalentPrimeIdeal; it is 64 for all three parameter sets.
- $\mathrm{MR}_{\mathrm{iter}}$ is the number of Miller–Rabin iterations used for pseudo-primality testing; it is 28, 63, and 144 for NGCC-1, NGCC-2, and NGCC-3, respectively.
- `cornacchia_prime_list` is the fixed list used by CornacchiaGeneral; it consists of 2 and the first 100 odd primes $\ell \equiv 1 \pmod 4$, and is the same for all three parameter sets.

For ease of reading, we will omit these auxiliary quantities as inputs and refer the reader

to Chapters 2 and 3 for their usage.

4.1.2 Hash Function

Following the construction in [BBC+25, BBC+26], we also need to define a hash function $\mathsf{H}_{\mathsf{PRISM}}$ that hashes onto the set of odd integers of length exactly $a-1$ bits. The hash function $\mathsf{H}_{\mathsf{PRISM}} : \mathbb{F}_{p^2} \times \{0,1\}^* \times \{0,1\}^{n_{\mathsf{salt}}} \to \{2^{a-1} + 2x + 1 | 0 \leq x < 2^{a-2}\}$ digests the j-invariant of a supersingular curve, a message msg and some $\mathsf{salt} \in \{0,1\}^{n_{\mathsf{salt}}}$, and returns an odd integer of length $a-1$ bits. More precisely $\mathsf{H}_{\mathsf{PRISM}}(j(E), \mathsf{msg}, \mathsf{salt})$ computes $h \leftarrow \mathsf{H}_{a-2}(j(E) \| \mathsf{msg} \| \mathsf{salt})$ and returns $q = 2^{a-1} + 2h + 1$.

4.2 Key Generation

The key generation procedure is described in QIMEN-PRISM.KeyGen. First, we sample a random ideal I_{sk} of prime norm N_{sk} using RandFixNormIdeal. Second, we compute the codomain of the isogeny $\phi_{\mathsf{sk}} : E_0 \to E_{\mathsf{pk}}$ corresponding to RandFixNormIdeal together with $\phi_{\mathsf{sk}}(P_0), \phi_{\mathsf{sk}}(Q_0)$ using IdealToIso. Then, we use TorsionBasisToHint to deterministically compute generators $P_{\mathsf{pk}}, Q_{\mathsf{pk}}$ of $E_{\mathsf{pk}}(\mathbb{F}_{p^2})[2^{a+2}]$ together with a hint $\mathsf{hint}_{\mathsf{pk}}$. Finally, compute the matrix M_{sk} sending $(\phi_{\mathsf{sk}}(P_0), \phi_{\mathsf{sk}}(Q_0))$ into $(P_{\mathsf{pk}}, Q_{\mathsf{pk}})$, i.e. the matrix mod 2^{a+2} given by

$$M_{\mathsf{sk}} = \begin{pmatrix} m_{11} & m_{12} \\ m_{21} & m_{22} \end{pmatrix}$$

such that $P_{\mathsf{pk}} = [m_{11}]\phi_{\mathsf{sk}}(P_0) + [m_{12}]\phi_{\mathsf{sk}}(Q_0)$ and $Q_{\mathsf{pk}} = [m_{21}]\phi_{\mathsf{sk}}(P_0) + [m_{22}]\phi_{\mathsf{sk}}(Q_0)$. This matrix is computed using ChangeOfBasis$_{2^{a+2}}$. The public key is set to be $(E_{\mathsf{pk}}, \mathsf{hint}_{\mathsf{pk}})$, and the secret key is $(E_{\mathsf{pk}}, I_{\mathsf{sk}}, M_{\mathsf{sk}})$.

4.3 Signing

The signing procedure is described in QIMEN-PRISM.Sign. Using the hash in Section 4.1.2, defining the signing step requires to specify the isogeny generation procedure, that is given in QIMEN-PRISM.GenIso. Parse pk as E_{pk} and sk as $(E_{\mathsf{pk}}, I_{\mathsf{sk}}, M_{\mathsf{sk}})$. The goal of the algorithm is to compute the codomain curve E_{sig} of an isogeny $\sigma : E_{\mathsf{pk}} \to E_{\mathsf{sig}}$ of degree $q(2^a - q)$ and $([q^{-1} \mod 2^{a+2}]\sigma(P_{\mathsf{pk}}), [q^{-1} \mod 2^{a+2}]\sigma(Q_{\mathsf{pk}}))$.

The first step is to generate an $\mathcal{O}_0$-ideal I_{chall} of norm $q(2^a - q)$: this is done using PrimeGenerator to get a prime q and the RandFixNormIdeal algorithm. The algorithm PrimeGenerator takes as inputs a curve E and a message msg: it generates a random salt by sampling a bit string of length n_{salt} uniformly from $\{0,1\}^{n_{\mathsf{salt}}}$ and applies $\mathsf{H}_{\mathsf{PRISM}}$ to $(E, \mathsf{msg}, \mathsf{salt})$ to obtain a q on which $\mathsf{PrimalityTest}$ is run on. The output is (q, salt). The ideal corresponding to the isogeny $\varphi = \sigma \circ \phi_{\mathsf{sk}} : E_0 \to E_{\mathsf{sig}}$ is given by $I = I_{\mathsf{sk}} \cap I_{\text{chall}}$, and using IdealToIso we can obtain a representation of φ and compute

Algorithm 17 QIMEN-PRISM.KeyGen(λ_c, λ_q)

Input: Target classical and quantum security parameters λ_c and λ_q.
Output: Secret key sk and public key pk.

while true **do**
  $I_{\mathsf{sk}} \leftarrow$ RandFixNormIdeal(N_{sk}, True)
  try
    $I_{\mathsf{sk}} \leftarrow$ RandomEquivalentPrimeIdeal(I_{sk})
    $E_{\mathsf{pk}}, \varphi_{\mathsf{sk}}(P_0), \varphi_{\mathsf{sk}}(Q_0) \leftarrow$ IdealToIso(I_{sk})
  catch
    continue
  $P_{\mathsf{pk}}, Q_{\mathsf{pk}}, \mathsf{hint}_{\mathsf{pk}} \leftarrow$ TorsionBasisToHint($E_{\mathsf{pk}}, a+2$)
  $M_{\mathsf{sk}} \leftarrow$ ChangeOfBasis$_{2^{a+2}}(E_{\mathsf{pk}}, (\varphi_{\mathsf{sk}}(P_0), \varphi_{\mathsf{sk}}(Q_0)), (P_{\mathsf{pk}}, Q_{\mathsf{pk}}))$
  $\mathsf{pk} \leftarrow (E_{\mathsf{pk}}, \mathsf{hint}_{\mathsf{pk}})$
  $\mathsf{sk} \leftarrow (E_{\mathsf{pk}}, \mathsf{hint}_{\mathsf{pk}}, I_{\mathsf{sk}}, M_{\mathsf{sk}})$
  return sk, pk

Algorithm 18 QIMEN-PRISM.GenIso(sk, q)

Input: Secret key $\mathsf{sk} = (E_{\mathsf{pk}}, I_{\mathsf{sk}}, M_{\mathsf{sk}})$ and prime number $q \in (2^{a-1}, 2^a)$.
Output: Signature curve E_{sig} and the points $(P_{\mathrm{sig}}, Q_{\mathrm{sig}})$.

$N \leftarrow q(2^a - q)$
$I_{\mathrm{chall}} \leftarrow$ RandFixNormIdeal (N, False)
$I \leftarrow$ IdealIntersection$(I_{\mathsf{sk}}, I_{\mathrm{chall}})$
$E_{\mathsf{sig}}, \varphi(P_0), \varphi(Q_0) \leftarrow$ IdealToIso(I)
$t \leftarrow q^{-1} \mod 2^{a+2}$
$P_{\mathrm{sig}} \leftarrow [t \cdot m_{11}]\varphi(P_0) + [t \cdot m_{12}]\varphi(Q_0)$
$Q_{\mathrm{sig}} \leftarrow [t \cdot m_{21}]\varphi(P_0) + [t \cdot m_{22}]\varphi(Q_0)$
return $(E_{\mathsf{sig}}, P_{\mathrm{sig}}, Q_{\mathrm{sig}})$

the images $\varphi(P_0), \varphi(Q_0)$. Setting E_{sig} as the codomain φ and $t = q^{-1} \mod 2^{a+2}$, we finally obtain

$$P_{\mathrm{sig}} = [t \cdot m_{11}]\varphi(P_0) + [t \cdot m_{12}]\varphi(Q_0), \quad Q_{\mathrm{sig}} = [t \cdot m_{21}]\varphi(P_0) + [t \cdot m_{22}]\varphi(Q_0).$$

4.4 Verification

We now define the verification algorithm described in QIMEN-PRISM.Verif. Parse the input interpolation isogeny iso as $(E_{\mathsf{sig}}, P_{\mathrm{sig}}, Q_{\mathrm{sig}})$, where $P_{\mathrm{sig}} = [q^{-1} \bmod 2^{a+2}]\,\sigma(P)$ and $Q_{\mathrm{sig}} = [q^{-1} \bmod 2^{a+2}]\,\sigma(Q)$. We recall that $E_{\mathsf{pk}}[2^{a+2}] = \langle P_{\mathsf{pk}}, Q_{\mathsf{pk}} \rangle$. Firstly, we call the hash function $\mathsf{H}_{\mathsf{PRISM}}$ on the inputs $(E_{\mathsf{pk}}, \mathsf{msg}, \mathsf{salt})$ and run PrimalityTest on its output q.

Algorithm 19 PrimeGenerator(E, msg)

Input: A curve E and a message msg.
Output: An integer q and salt.

1: **while** True **do**
2: $\quad \mathsf{salt} \xleftarrow{\$} \{0,1\}^{n_{\mathsf{salt}}}$
3: $\quad q \leftarrow \mathsf{H}_{\mathsf{PRISM}}(E, \mathsf{msg}, \mathsf{salt})$
4: $\quad$ **if** PrimalityTest(q) **then**
5: $\quad\quad$ Break
6: **return** (q, salt)

Algorithm 20 QIMEN-PRISM.Sign$(\mathsf{sk}, \mathsf{msg})$

Input: Secret key $\mathsf{sk} = (E_{\mathsf{pk}}, I_{\mathsf{sk}}, M_{\mathsf{sk}})$ and message msg.
Output: Signature curve E_{sig} and the points $(P_{\mathrm{sig}}, Q_{\mathrm{sig}})$, together with the salt salt.

1: $(q, \mathsf{salt}) \leftarrow$ PrimeGenerator$(E_{\mathsf{pk}}, \mathsf{msg})$
2: $E_{\mathsf{sig}}, P_{\mathrm{sig}}, Q_{\mathrm{sig}} \leftarrow$ QIMEN-PRISM.GenIso$(\mathsf{sk}, \mathsf{pk}, q)$
3: $P'_{\mathrm{sig}}, Q'_{\mathrm{sig}}, \mathsf{hint}_{\mathrm{sig}} \leftarrow$ TorsionBasisToHint$(E_{\mathsf{sig}}, a+2)$
4: $M_{\mathrm{sig}} \leftarrow$ ChangeOfBasis$_{2^{a+2}}(E_{\mathsf{sig}}, (P'_{\mathrm{sig}}, Q'_{\mathrm{sig}}), (P_{\mathrm{sig}}, Q_{\mathrm{sig}}))$
5: $\sigma \leftarrow (E_{\mathsf{sig}}, M_{\mathrm{sig}}, \mathsf{hint}_{\mathrm{sig}})$
6: **return** (σ, salt)

Then, we have to check that these points interpolate an isogeny $\sigma : E_{\mathsf{pk}} \to E_{\mathsf{sig}}$ of degree $q(2^a - q)$, and to do it we use the function CheckIsogeny$(E_{\mathsf{pk}}, P_{\mathsf{pk}}, Q_{\mathsf{pk}}, E_{\mathsf{sig}}, P_{\mathrm{sig}}, Q_{\mathrm{sig}}, a, q)$, that returns true if and only if there exists an isogeny $\varphi : E_{\mathsf{pk}} \to E_{\mathsf{sig}}$ of degree $q(2^a - q)$ such that $[q]P_{\mathrm{sig}} = \varphi(P_{\mathsf{pk}})$ and $[q]Q_{\mathrm{sig}} = \varphi(Q_{\mathsf{pk}})$.

4.5 Binary format

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

- Elements of $\mathbb{F}_p$ are encoded as unsigned integers between 0 and $p-1$, in little-endian, using `FP_ENCODED_BYTES` bytes
- Elements of $\mathbb{F}_{p^2}$ are encoded by concatenating the encoding of the real part with the encoding of the imaginary part, both encoded as $\mathbb{F}_p$ elements
- Integers in $\mathbb{Z}$ are encoded in little-endian two's complement representation; the number of bytes is fixed on a per-instance basis by the specification.
- Elliptic curves are encoded by their Montgomery coefficient $A \in \mathbb{F}_{p^2}$.
- Ideals I are encoded by concatenating the encoding of $\mathrm{nrd}(I)$, which is a `FP_ENCODED_BYTES`-

Algorithm 21 QIMEN-PRISM.Verif($\mathsf{pk}, \mathsf{msg}, \sigma, \mathsf{salt}$)

Input: Public key $\mathsf{pk} = (E_{\mathsf{pk}}, \mathsf{hint}_{\mathsf{pk}})$, a message msg, and a signature (σ, salt).
Output: accept or reject.

$q \leftarrow \mathsf{H}_{\mathsf{PRISM}}(E_{\mathsf{pk}}, \mathsf{msg}, \mathsf{salt})$
if $\mathsf{PrimalityTest}(q) = \mathsf{False}$ **then**
    return reject
$(P_{\mathsf{pk}}, Q_{\mathsf{pk}}) \leftarrow$ TorsionBasisFromHint$(E_{\mathsf{pk}}, \mathsf{hint}_{\mathsf{pk}}, a+2)$
Parse σ as $E_{\mathsf{sig}}, M_{\mathrm{sig}}, \mathsf{hint}_{\mathrm{sig}}$
$P_{\mathrm{sig}}, Q_{\mathrm{sig}} \leftarrow$ TorsionBasisFromHint$(E_{\mathsf{sig}}, \mathsf{hint}_{\mathrm{sig}}, a+2)$
$P_{\mathrm{sig}}, Q_{\mathrm{sig}} \leftarrow M_{\mathrm{sig}} \cdot (P_{\mathrm{sig}}, Q_{\mathrm{sig}})$
if CheckIsogeny$(E_{\mathsf{pk}}, P_{\mathsf{pk}}, Q_{\mathsf{pk}}, E_{\mathsf{sig}}, P_{\mathrm{sig}}, Q_{\mathrm{sig}}, a, q)$ **then**
    return accept
else
    return reject

byte integer, with the encoding of the quaternion output by IdealGenerator(I). This quaternion is encoded by the concatenation of four `FP_ENCODED_BYTES`-byte integers, denoting the integer coefficients in the $1, i, j, k$-basis.

- 2×2 integer matrices $\left(\begin{smallmatrix} a & b \\ c & d \end{smallmatrix}\right)$ are encoded by the concatenation of the encoding of a, b, c and d.
- Hints consist of two values (h_A, h); the quantity h_A is either zero or one, whereas h is a 7-bit integer. A hint is then encoded in one byte, where the least significant bit represents the flag h_A.
- The salt salt is n_{salt} bits long, then it is encoded in $n_{\mathsf{salt}}/8$ bytes.

Public keys. The public key is encoded by concatenating the encodings of E_{pk} from Step 5 of QIMEN-PRISM.KeyGen together with the hint $\mathsf{hint}_{\mathsf{pk}}$ from Step 8 of QIMEN-PRISM.KeyGen.

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

- the encoding of the public key pk;
- the ideal I_{sk} from Step 4 of QIMEN-PRISM.KeyGen;
- the 2×2 matrix M_{sk} from Step 9 of QIMEN-PRISM.KeyGen.

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

- the Montgomery coefficient of E_{sig} from Step 2 of QIMEN-PRISM.Sign;

- the 2×2 matrix M_{sig} from Step 4 of QIMEN-PRISM.Sign;
- the hint $\mathsf{hint}_{\mathrm{sig}}$ from Step 3 of QIMEN-PRISM.Sign;
- the salt salt from Step 1 of QIMEN-PRISM.Sign.

CHAPTER 5

Parameter sets

NGCC-1.	$p = 69 \cdot 2^{313} - 1$,	$a = 224$,	$n_{\mathsf{salt}} = 232$,	$\mathrm{MR}_{\mathrm{iter}} = 28$,
NGCC-2.	$p = 27 \cdot 2^{500} - 1$,	$a = 320$,	$n_{\mathsf{salt}} = 336$,	$\mathrm{MR}_{\mathrm{iter}} = 63$,
NGCC-3.	$p = 15 \cdot 2^{1004} - 1$,	$a = 576$,	$n_{\mathsf{salt}} = 592$,	$\mathrm{MR}_{\mathrm{iter}} = 144$.

This specification defines two approved instantiations of the auxiliary hash function

$$\mathsf{H}_{a-2} : \{0,1\}^* \to [0, 2^{a-2})$$

from an extendable hash function: the SHAKE256 instantiation and the pseudoXOF instantiation, where pseudoXOF is a SM3-based function provided by NGCC (GB/T 32918.4-2016, Section 5.4.3). As per the submission regulation, pseudoXOF is only used for the examination purpose and shall NOT be considered as a secure XOF function.

In both instantiations, the domain-separation prefix HPRISM shall be used.

SHAKE256 instantiation.

$$\mathsf{H}_{a-2}(x) = \mathrm{trunc}_{a-2}(\mathsf{SHAKE256}(\mathsf{HPRISM} \| x)),$$

where trunc_{a-2} denotes truncation to the first $a-2$ output bits, interpreted as a non-negative integer in $[0, 2^{a-2})$. The hash function of Section 4.1.2 is then instantiated as

$$\mathsf{H}_{\mathsf{PRISM}}(j(E), \mathsf{msg}, \mathsf{salt}) = 2^{a-1} + 2\mathsf{H}_{a-2}(j(E) \| \mathsf{msg} \| \mathsf{salt}) + 1.$$

pseudoXOF instantiation.

$$\mathsf{H}_{a-2}(x) = \mathrm{trunc}_{a-2}(\mathsf{pseudoXOF}(\mathsf{HPRISM} \| x)),$$

where trunc_{a-2} denotes truncation to the first $a-2$ output bits, interpreted as a non-negative integer in $[0, 2^{a-2})$. The hash function of Section 4.1.2 is then instantiated as

$$\mathsf{H}_{\mathsf{PRISM}}(j(E), \mathsf{msg}, \mathsf{salt}) = 2^{a-1} + 2\mathsf{H}_{a-2}(j(E) \| \mathsf{msg} \| \mathsf{salt}) + 1.$$

An implementation shall use one of these two instantiations consistently. If additional functions are derived from the same XOF, they shall use distinct domain-separation prefixes.

CHAPTER 6

Test vectors

We provide known-answer test vectors for all QIMEN-PRISM parameter sets in the folder `Test_Vector/` of the submission.

For QIMEN-PRISM, the files are `KAT_SIG_NGCC-1.txt`, `KAT_SIG_NGCC-2.txt`, and `KAT_SIG_NGCC-3.txt`; each record contains a deterministic seed, a public key, a secret key, a message, and the resulting signature. The corresponding KAT generation programs are included in `Implementations/` and regenerate the package-level test-vector files with the same build options.

CHAPTER 7

Performance analysis

We provide both a reference implementation in C and an optimized implementation incorporating assembly-optimized finite-field arithmetic in the submission package. Both implementations are derived from the PRISM-salt implementation.[1] Note that this codebase builds upon the SQIsign library [AAA+25].

Several submodules in the SQIsign library provide low-level building blocks, which are untouched in the implementation of QIMEN-PRISM. These are:

- `hd`: module to compute $(2, 2)$-isogenies in the theta model.
- `id2iso`: module to evaluate the two-dimensional isogenies arising from ideals.
- `quaternion`: quaternion computation module supporting both GMP-driven arbitrary precision and DPE-based floating-point arithmetic.

The reference implementation comes with the following CMake build options:

- `CMAKE_BUILD_TYPE=Release` selects the optimized build configuration.
- `ENABLE_SIGN=ON` builds the signature implementation.
- `PRISM_NGCC_XOF_BACKEND` selects the NGCC XOF backend; supported values are `shake` and `sm3`.
- `ENABLE_NGCC_PRISM=ON` builds the NGCC KAT generation targets. The KAT output directory is selected by `NGCC_PRISM_KAT_OUTPUT_DIR`.

When `ENABLE_TESTS=ON`, the NGCC KAT generation programs are registered as CTest entries. We implement both SHAKE and SM3 as NGCC XOF backends: SHAKE is a standard and widely used XOF, while the NGCC official sample code provides a pseudo-XOF construction based on SM3.

7.1 Key and signature sizes

Table 7.1 lists the public key, secret key, and signature sizes per parameter set.

[1] https://github.com/mariascrs/PRISM_v2

Table 7.1: QIMEN-PRISM key and signature sizes in bytes for each security level.

Parameter set	Public key	Secret key	Signature
NGCC-1	81	441	225
NGCC-2	129	701	334
NGCC-3	257	1401	622

7.2 Performance evaluation

This section gives the performance of both the reference implementation and optimized implementation of QIMEN-PRISM in terms of CPU cycles.

- **Platforms:** WSL on `x86_64` machines: Intel(R) Core(TM) Ultra 9 275HX CPU at 2.70 GHz and 32 GB RAM.
- **Compiler and build system:** GCC 13.3.0 with CMake in `Release` mode.
- **Dependencies:** the implementation links against GMP for multiprecision integer arithmetic. SHAKE/FIPS202, AES/CTR-DRBG, SM3, and the `randombytes` routines are bundled in the codebase rather than provided by external cryptographic libraries. The finite field arithmetic in C is generated by automated script in [Sco24]. The codebase builds on the SQIsign implementation repository [AAA+25].
- **Implementations:** the reference implementation was built from `Implementations/`, while the assembly-optimized was built from `Optimized_Implementation/`.
- **Hash/XOF backend:** following the NGCC submission requirements, both builds used the SM3 backend by configuring `PRISM_NGCC_XOF_BACKEND=sm3`, which sets `PRISM_XOF_BACKEND=2`.
- **Compilation settings:** the effective C flags included `-O3`, `-DNDEBUG`, `-std=c11`, `-fvisibility=hidden`, and `-funroll-loops`; both builds defined `RADIX_64`, `TARGET_AMD64`, and `TARGET_OS_UNIX`.
- **Parameter sets:** the reported benchmarks use NGCC-1, NGCC-2, and NGCC-3.
- **Measurements:** each reported signature operation is averaged over 300 runs.

7.2.1 Reference Implementation

Table 7.2 show the current reference implementation performance using SM3.

Table 7.2: Reference QIMEN-PRISM signature performance using SM3 as the XOF, in 10^6 CPU cycles, measured on an Intel(R) Core(TM) Ultra 9 275HX CPU, averaged over 300 runs.

Parameter set	KeyGen	Sign	Verify
NGCC-1	66.95	78.14	14.23
NGCC-2	129.66	166.12	41.33
NGCC-3	684.67	969.23	231.23

Table 7.3: Reference QIMEN-PRISM signature throughput using SM3 as the XOF, in operations per second (ops/s), measured on an Intel(R) Core(TM) Ultra 9 275HX CPU, averaged over 300 runs.

Parameter set	KeyGen	Sign	Verify
NGCC-1	42.7	40.8	204.3
NGCC-2	22.3	20.1	71.7
NGCC-3	4.3	3.8	12.4

7.2.2 Optimized Implementation

The optimized build used additional flag

```
PRISM_BUILD_TYPE=broadwell
```

enabling x86-64 BMI2/ADX assembly field arithmetic for NGCC-1 and NGCC-2, while NGCC-3 uses the same portable C core. Table 7.4 shows the assembly-optimized implementation performance. We are currently unable to provide the assembly-optimization for the NGCC-3.

Table 7.4: Assembly-optimized QIMEN-PRISM signature performance using SM3 as the XOF, in 10^6 CPU cycles, measured on an Intel(R) Core(TM) Ultra 9 275HX CPU, averaged over 300 runs.

Parameter set	KeyGen	Sign	Verify
NGCC-1	57.32	66.27	9.71
NGCC-2	90.24	122.52	24.57

Table 7.5: Optimized QIMEN-PRISM signature throughput using SM3 as the XOF, in operations per second (ops/s), measured on an Intel(R) Core(TM) Ultra 9 275HX CPU, averaged over 300 runs.

Parameter set	**KeyGen**	**Sign**	**Verify**
NGCC-1	46.2	44.4	269.0
NGCC-2	31.6	27.2	118.7

CHAPTER 8

Security

This chapter discusses the main security aspects of the QIMEN-PRISM digital signature, following the analyses performed on salt-PRISM [BBC+26]. First, in Section 8.1 we discuss the mathematical hardness assumptions at the foundations of QIMEN-PRISM. In Section 8.2, we state a theoretical result proving the security of the protocol in the quantum random oracle model, under these hardness assumptions. Then, in Section 8.3, we discuss further, more ad-hoc attacks on the scheme and their respective complexities. This includes key recovery, direct attacks on the hash function, and a brief discussion of non-resignability.

8.1 Hard problems

8.1.1 The endomorphism ring problem

The endomorphism ring problem is the common foundation for all of (supersingular) isogeny-based cryptography, including QIMEN-PRISM, and consequently its hardness forms an upper bound for the hardness of breaking this submission. As discussed in Section 3.3 the endomorphism ring of a supersingular elliptic curve E over $\mathbb{F}_{p^2}$ is isomorphic to a maximal order $\mathcal{O} \subset \mathcal{B}_{p,\infty}$. The endomorphism ring problem asks to compute this order explicitly.

Problem 8.1.1 (Endomorphism ring problem)**.** Given a supersingular elliptic curve $E/\mathbb{F}_{p^2}$, return $b_1, b_2, b_3 \in \mathcal{B}_{p,\infty}$ and efficient representations of endomorphisms $\beta_1, \beta_2, \beta_3$ such that the $\mathbb{Z}$-linear map from $\mathcal{O} := \langle 1, b_1, b_2, b_3 \rangle_{\mathbb{Z}}$ to $\mathrm{End}(E)$ defined by

$$1 \mapsto \mathrm{Id}, \qquad b_1 \mapsto \beta_1, \qquad b_2 \mapsto \beta_2, \qquad b_3 \mapsto \beta_3$$

is an isomorphism of rings.

Here an efficient representation of an isogeny $\varphi : E \to E'$ between two supersingular elliptic curves over $\mathbb{F}_{p^2}$ is thought of as an evaluation algorithm that runs in time polynomial in $\log p$, $\deg \varphi$, and the extension degree k of the defining field of the input point $P \in E(\mathbb{F}_{p^{2k}})$.

Several variants of the endomorphism ring problem have been studied, such as the mere computation of b_1, b_2, b_3 for which the order generated by $1, b_1, b_2, b_3$ admits an isomorphism to $\mathrm{End}(E)$; this is sometimes called the maximal order problem. All these variants were shown to be polynomial-time equivalent [Wes22]. The fastest classical algorithms for Problem 8.1.1 run in time $\tilde{O}(p^{1/2})$ [DG16]. These algorithms can be Groverized into quantum methods running in time $\tilde{O}(p^{1/4})$ [Gro96, BJS14]. In both cases, the methods can be executed with low memory requirements. Thus, to achieve a quantum security level of λ_q bits, it is required to use a prime p of at least about $4\lambda_q$ bits.

8.1.2 Large prime-degree isogeny problem

The security of QIMEN-PRISM hinges on the hardness of computing an (efficient representation of an) isogeny of large prescribed degree $q \neq p$ from a given supersingular elliptic curve $E/\mathbb{F}_{p^2}$ with $\#E(\mathbb{F}_{p^2}) = (p+1)^2$. This problem can always be solved in time $\tilde{O}(\max\{k^2, k\sqrt{q}\})$ with $k \mid q-1$ the smallest integer such that $(-p)^k \equiv 1 \bmod q$ [NO25, Lemma 3], with the square-root-Vélu algorithm from [BDLS20] as the main ingredient. The quantity k equals the extension degree over which a point of order q can be found. The following variant of the problem better captures the EUF-CMA attack model, where one has access to arbitrarily many large-prime degree isogenies prior to the attack:

Problem 8.1.2 (Large Prime-Degree Isogeny problem, LPDI)**.** Let $\{q_i\}_{i\in[N]}$ and $\bar{q}$ be $N+1$ primes sampled independently and uniformly at random from the set Primes_a of a-bit prime numbers. Given a uniformly random element E of the set Ell_p of supersingular elliptic curves over $\mathbb{F}_{p^2}$, a set of N isogenies $\{\phi_i : E \to E_i\}_{i\in[N]}$ of degree $q_i(2^a - q_i)$, where ϕ_i is sampled uniformly at random among the isogenies of degree $q_i(2^a - q_i)$, and the prime $\bar{q}$, compute an (efficient representation of an) isogeny of degree $\bar{q}(2^a - \bar{q})$.

If the endomorphism ring of E is known then the problem can be solved in polynomial time: this is basically how the trapdoor in QIMEN-PRISM works. It is believed, see [BBC+26, DLRW24], that knowing random large prime degree isogenies from E has no impact on the difficulty of computing its endomorphism ring.

One expects $k \approx 2^a$ in which case the cost computing a q-isogeny becomes $\tilde{O}(2^{3a/2})$, but k may exceptionally take smaller values in which case the attack becomes faster. For instance, assuming $k = 1$ the complexity becomes $\tilde{O}(2^{a/2})$. In [NO25], the probability that $k \ll 2^a$ was studied in detail and used for a forgery attack, a priori on PRISM-id but it also applies to QIMEN-PRISM, that runs in time $\tilde{O}(2^{6a/7})$. As suggested in [BBC+26, Remark 8] such attacks can be avoided by sampling $q_i, \bar{q}$ from the subset of *safe primes*, i.e., for which $(q-1)/2$ is also a prime number. Safe primes are prevalent: they occur with frequency in the order of $1/a^2$, so our choice $a = n = \lambda_c + 64$, made to mitigate attacks on the hash function, is more than enough for there to exist enough valid primes of bit length a. But in our parameter range we do not need to implement this countermeasure, which would increase the signing time. Indeed, while for the largest classical security

level $\lambda_c = 512$ the ratio $6(512+64)/7 \approx 493.71$ seemingly falls short, the gap is amply compensated by the hidden constant and logarithmic factors in $\tilde{O}(2^{6a/7})$, which come from resultant computations of polynomials of degree about $2^{a/2}$, defined over extensions of $\mathbb{F}_{p^2}$ of degree about $2^{2a/7}$.

Finally, given the current state of the art, no significant quantum speed-up appears to be achievable.

8.2 Theoretical security

Definition 8.2.1. For any probabilistic polynomial-time adversary $\mathcal{A}$, we define $\mathsf{Adv}^{p,N}_{\mathsf{LDPI}}(\mathcal{A})$ to be the advantage of $\mathcal{A}$ in solving a LDPI instance with prime p and N given isogenies, that is

$$\mathsf{Adv}^{\mathsf{LDPI}}_{p,N}(\mathcal{A}) := \Pr\left[\begin{array}{l|l} \bar{\phi} : E \to \bar{E} \text{ is an isogeny} & \{q_i\}_{i\in[N]}, \bar{q} \xleftarrow{\$} \mathsf{Primes}_a^{N+1}; \\ \text{of degree } \bar{q}(2^a - \bar{q}) & E \xleftarrow{\$} \mathsf{Ell}_p; \\ & \{\phi_i : E \to E_i\}_{i\in[N]} \text{ sampled u.a.r.} \\ & \text{such that } \deg(\phi_i) = q_i(2^a - q_i); \\ & \bar{\phi} \leftarrow \mathcal{A}\Big(E, \{\phi_i : E \to E_i\}_{i\in[N]}, \bar{q}\Big) \end{array}\right].$$

The QIMEN-PRISM protocol satisfies the standard security notion for signatures (EUF-CMA) assuming the hardness of the LDPI problem.

Theorem 8.2.2 ([BBC+25, Theorem 2])**.** *Let $\mathcal{A}$ be a PPT adversary against the* EUF-CMA *security of QIMEN-PRISM with prime p making at most N_{sign} signing queries and N_{H} (quantum) queries to the random oracle $\mathsf{H} : \mathsf{Ell}_p \times \{0,1\}^* \times \{0,1\}^{n_{\mathsf{salt}}} \to \{0,1\}^a$. Let $\gamma_a = \#\mathsf{Primes}_a/2^a$. In the quantum random oracle model, there exists an adversary $\mathcal{B}$ against the LDPI problem for $N = N_{\mathsf{sign}}$ such that*

$$\mathsf{Adv}^{\mathsf{EUF\text{-}CMA}}_{\mathsf{salt\text{-}PRISM}}(\mathcal{A}) \le (2N_{\mathsf{H}}+1)^2 \mathsf{Adv}^{\mathit{LDPI}}_{p,N_{\mathsf{sign}}}(\mathcal{B}) + \frac{N_{\mathsf{sign}}(N_{\mathsf{sign}}+N_{\mathsf{H}}+2)^2}{2^{a-3}} + 3N_{\mathsf{sign}}\sqrt{\frac{N_{\mathsf{H}}+N_{\mathsf{sign}}+1}{\gamma_a 2^{n_{\mathsf{salt}}}}}.$$

8.3 Practical security

8.3.1 Key recovery

The secret key in QIMEN-PRISM is mainly given by an ideal I_{sk} corresponding to an isogeny $\phi_{\mathsf{sk}} : E_0 \to E_{\mathsf{pk}}$ of degree N_{sk}. A secret matrix M_{sk} is also part of the secret key, however, it is directly computed from the secret ideal I_{sk} and only serves to efficiently represent ϕ_{sk}.

This ideal I_{sk} is sampled uniformly at random among the $N_{\mathsf{sk}}+1$ left $\mathcal{O}_0$-ideals of norm N_{sk}. Since N_{sk} is the smallest prime larger than $2^{4\lambda_c}$, an exhaustive search among all $\mathcal{O}_0$-ideals of norm N_{sk} is impractical. Currently, no better searching methods are known.

Following the discussion in [DLRW24, §4], the prime N_{sk} is large enough to guarantee that the public elliptic curve E_{pk} is indistinguishable from a random curve. It is easy to see that the problem of computing the endomorphism ring of a random supersingular elliptic curve reduces to the problem of recovering the associated secret key, using [DLRW24, Algorithm 8] for instance. If it is indeed true that knowing random large prime degree isogenies from E does not make the endomorphism ring problem any easier, then key recovery is equivalent to Problem 8.1.1.

8.3.2 Signature forgery

Collision attack:

In [BBC+26, §4.4], it is argued that without salt, the parameter a is required to meet $2^{\lambda_c} \leq 2^{(a-2)/2}\sqrt{a}$ to ensure that signature forgery via collision attacks on the hash function $\mathsf{H}_{\mathsf{PRISM}}$ is infeasible within $o(2^{\lambda_c})$ operations; this leads to the estimate that a should have bit length at least about $2\lambda_c - \log \lambda_c$. Concretely, the collision attack looks offline for a collision $\mathsf{H}_{\mathsf{PRISM}}(j(E_{\mathsf{pk}}), \mathsf{msg}_1, \mathsf{counter}_1) = \mathsf{H}_{\mathsf{PRISM}}(j(E_{\mathsf{pk}}), \mathsf{msg}_2, \mathsf{counter}_2)$ with $\mathsf{msg}_1 \neq \mathsf{msg}_2$ and where the $\mathsf{counter}_i$'s are minimal such that the result is in Primes_a, using repeated calls to $\mathsf{H}_{\mathsf{PRISM}}$. It then calls the signing oracle once, on input msg_1, to obtain a valid signature for msg_2.

Following [BBC+26, §4.4], by replacing the counter with random salt, the impact of offline collision attacks is strongly reduced, which allows one to consider a smaller parameter a, resulting in a more practical scheme that reaches the same security level. While this is already taken into account in the security proof provided by Theorem 8.2.2 in the QROM, let us discuss the collision attacks that remain available in the salted setting. This will also clarify the choice of the parameters a and n_{salt} of QIMEN-PRISM made in Chapter 5.

- Offline-then-online: A naive adaptation of the above attack looks for random pairs of messages and salts until getting a prime-valued collision $\mathsf{H}_{\mathsf{PRISM}}(j(E_{\mathsf{pk}}), \mathsf{msg}_1, \mathsf{salt}_1) = \mathsf{H}_{\mathsf{PRISM}}(j(E_{\mathsf{pk}}), \mathsf{msg}_2, \mathsf{salt}_2)$, for $\mathsf{msg}_1 \neq \mathsf{msg}_2$. However, turning this into a signature forgery with a single call to the signing oracle is no longer possible: the oracle returns a signature $(\sigma, \mathsf{salt}'_1)$ for msg_1, for some random salt salt'_1, and with overwhelming probability $\mathsf{H}_{\mathsf{PRISM}}(j(E_{\mathsf{pk}}), \mathsf{msg}_1, \mathsf{salt}'_1) \neq \mathsf{H}_{\mathsf{PRISM}}(j(E_{\mathsf{pk}}), \mathsf{msg}_2, \mathsf{salt}_2)$ so that $(\sigma, \mathsf{salt}_2)$ is not a valid signature for msg_2.
  However, using multiple calls to the signing oracle one can hope that eventually a signature with $\mathsf{salt}'_1 = \mathsf{salt}_1$ is returned. If we write N_{sign} for the number of calls, then we expect this to succeed with probability N_{sign}/h, with h the number of $\mathsf{salt} \in \{0,1\}^{n_{\mathsf{salt}}}$ such that $\mathsf{H}_{\mathsf{PRISM}}(j(E_{\mathsf{pk}}), \mathsf{msg}_1, \mathsf{salt}) \in \mathsf{Primes}_a$. So it is natural to estimate this as $2^{n_{\mathsf{salt}}}\gamma_a$. Thus, to guarantee that this probability is negligible, i.e., to have

$\frac{N_{\mathsf{sign}}}{2^{n_{\mathsf{salt}}}\gamma_a} \leq 2^{-\lambda_c}$, we need $n_{\mathsf{salt}} \geq \lambda_c + \log_2(N_{\mathsf{sign}}) - \log_2(\gamma_a)$.

- Online-then-offline: The size of a is informed by the attack scenario where one first calls the signing oracle to get N_{sign} signatures $(\sigma_i, \mathsf{salt}_i)$ on N_{sign} messages msg_i. Then the attacker fixes any message msg' and samples random salts salt' until getting a collision $\mathsf{H}_{\mathsf{PRISM}}(j(E_{\mathsf{pk}}), \mathsf{msg}_i, \mathsf{salt}_i) = \mathsf{H}_{\mathsf{PRISM}}(j(E_{\mathsf{pk}}), \mathsf{msg}', \mathsf{salt}')$, so that $(\sigma_i, \mathsf{salt}_i)$ is a valid signature for msg'. The probability that salt' realizes a collision is about $N_{\mathsf{sign}}/2^a$. To guarantee that this probability is negligible we need $a \geq \lambda_c + \log_2(N_{\mathsf{sign}})$.

These bounds are also justified by the advantage

$$\frac{\mathsf{Adv}^{\mathsf{EUF\text{-}CMA}}_{\mathsf{salt\text{-}PRISM}}(\mathcal{A})}{N_{\mathsf{H}} + N_{\mathsf{sign}}} \leq \mathsf{Adv}^{\mathsf{LDPI}}_{p,N_{\mathsf{sign}}}(\mathcal{B}) + \frac{N_{\mathsf{sign}}}{2^a} + \frac{N_{\mathsf{sign}}}{\gamma_a 2^{n_{\mathsf{salt}}-1}},$$

established in [BBC+26, Theorem 2] in the ROM; here the cost of the attack is lower bounded as $N_{\mathsf{H}} + N_{\mathsf{sign}}$. The choice for working in the ROM rather than the QROM is justified by the fact that the quadratic loss obtained in the QROM is believed to be an artifact of the proof techniques, as is common in the hash-and-sign paradigm. Hence, for λ_c bits of classical security and with $N_{\mathsf{sign}} = 2^{64}$, setting $a = \lambda_c + 64$ and $n_{\mathsf{salt}} = \lambda_c + 64 + \log_2(a)$ is sufficient.

Finally, one can notice that the condition on a is easy to satisfy. In fact, by the previous discussion about the difficulty of Problem 8.1.1, the prime p is already required to have about $4\lambda_q$ bits, which is more than $2\lambda_c$ bits. Hence, by construction, there are enough bits of accessible 2-torsion to exceed the required a bits.

Hash to pseudo-prime attack:

In QIMEN-PRISM, the algorithm PrimeGenerator relies on the Miller–Rabin method to test for primality of the hashes output by $\mathsf{H}_{\mathsf{PRISM}}$. The Miller–Rabin algorithm has a non-zero probability of error, meaning that an integer may be incorrectly classified as prime. This probability decreases as the number of iterations of Miller–Rabin, denoted $\mathrm{MR}_{\mathrm{iter}}$, grows. As shown in [CGMT26], it is possible for an adversary to exploit such errors to forge signatures.

In order to counter this kind of attack, we fix the parameter $\mathrm{MR}_{\mathrm{iter}}$ such that the error probability of the Miller–Rabin test is smaller than $2^{-\lambda_c}$. This choice is based on the average case analysis done in [DLP93]. In particular, [DLP93, Theorem 6] provides the following upper bound on the error probability $p_{k,t}$ of the Miller–Rabin test for random odd k-bit inputs after t iterations, when $k \geq 21$ and $t \geq k/9$:

$$p_{k,t} < \frac{7}{20} k 2^{-5t} + \frac{1}{7} k^{15/4} 2^{-k/2-2t} + 12k 2^{-k/4-3t}.$$

For the three security levels 128, 256, and 512, the corresponding bit lengths of the integers tested by the Miller–Rabin method are given by $a = 224$, 320 and 576. Hence, we set

$\mathrm{MR}_{\mathrm{iter}} = 28$, 63, and 144, respectively, so that the resulting error probabilities satisfy $p_{224,28} \leq 2^{-128}$, $p_{320,63} \leq 2^{-256}$, and $p_{576,144} \leq 2^{-544}$.

Note that the above analysis focuses on classical security, as no methods are currently known to speed up the search for a pseudo-prime hash using a quantum computer.

8.3.3 Non Re-signability

As in [BBC+26], we achieve the non re-signability property defined in [CDF+21] by including the domain isogeny E_{pk} in the argument of the hashing, which is almost free of cost. This binds each signature to its signer's public key. Hence, this prevents adversaries from re-using a valid signature verifying under one public key to forge a valid signature under a different public key.

8.4 Parameter security assessment

Table 8.1 reports the cost of the best known classical and quantum attacks on QIMEN-PRISM, instantiated with the parameters of Chapter 5, confirming that the target security levels are achieved. We use the following notations in the table:

⋆: The reported complexity accounts only for the first step of the attack, namely finding a hash output landing on a pseudo-prime. Completing the attack requires additional costly operations that significantly increase the overall computational cost. We refer to [CGMT26] for more details.

†: The gap between the complexity estimate and the target security level is offset by the cost of arithmetic over the finite field with $\mathbb{F}_p$, which was not accounted for in the asymptotic estimate.

‡: The 18-bit shortfall between the complexity estimate and the target security level is accounted for by the hidden constant and logarithmic factors inherent to the $\tilde{O}$ as discussed in Section 8.1.2.

§: The exponents are rounded up, e.g. we write $2^{79.7\ldots}$ as 2^{80}.

Table 8.1: Choice of parameters and attack complexities for QIMEN-PRISM

		NGCC-1	**NGCC-2**	**NGCC-3**
		Parameters		
	p	$69 \cdot 2^{313} - 1$	$27 \cdot 2^{500} - 1$	$15 \cdot 2^{1004} - 1$
	a	224	320	576
	n_{salt}	232	336	592
	$\mathrm{MR}_{\mathrm{iter}}$	28	63	144
	N_{sign}	2^{64}	2^{64}	2^{64}
Required classical security strength level		128	256	512
Classical attack	**Complexity class**	**Complexity estimate**§		
On Problem 8.1.1	$\tilde{O}(p^{1/2})$	2^{160}	$2^{252\dagger}$	$2^{504\dagger}$
On Problem 8.1.2	$\tilde{O}(2^{6a/7})$	2^{192}	2^{274}	$2^{494\ddagger}$
Offline–then–online	$\tilde{O}(\frac{2^{n_{\mathsf{salt}}\gamma_a}}{N_{\mathsf{sign}}})$	2^{160}	2^{263}	2^{518}
Online–then–offline	$\tilde{O}(\frac{2^a}{N_{\mathsf{sign}}})$	2^{160}	2^{256}	2^{512}
Hash to pseudo-prime		$2^{128\star}$	$2^{256\star}$	$2^{544\star}$
Required quantum security strength level		80	128	256
Quantum attack	**Complexity class**	**Complexity estimate**§		
On Problem 8.1.1	$\tilde{O}(p^{1/4})$	2^{80}	$2^{126\dagger}$	$2^{252\dagger}$

CHAPTER 9

Failure analysis

The role of this chapter is to list and analyze possible failure cases of various algorithms involved in QIMEN-PRISM. Specifically, this chapter discusses heuristics behind the failure analysis, calculates failure probabilities for each algorithm that may fail for all three security levels, and along the way, explains some parameter choices.

In QIMEN-PRISM, failures are handled differently depending on the protocol stage. During key generation, the initial call to RandFixNormIdeal uses the `prime=true` branch. Later exceptions raised by RandomEquivalentPrimeIdeal or by IdealToIso are caught in QIMEN-PRISM.KeyGen; when such an exception is caught, key generation restarts. During signing, exceptions raised by QIMEN-PRISM.GenIso are propagated as signing failures. During verification, exceptions raised inside CheckIsogeny are caught and converted into false, which causes QIMEN-PRISM.Verif to return reject.

The possible sources of failure are as follows:

- RandomEquivalentPrimeIdeal may raise an exception if its search does not find an equivalent ideal of prime reduced norm within the prescribed search space.
- GeneralizedRepresentInteger may raise an exception if its bounded search does not find a quaternion element of the target norm. This exception is propagated by RandFixNormIdeal when the latter is called with `prime=false`.
- Qlapoti may raise an exception if its bounded search over generators does not find suitable elements β_1, β_2.
- IdealToIso may fail by propagating an exception raised by Qlapoti or by Isogeny22Chain.
- CheckIsogeny catches any exception raised by its call to Isogeny22Chain and returns false.

In the following sections, we analyze the failure probabilities of RandomEquivalentPrimeIdeal, GeneralizedRepresentInteger, Qlapoti, and Isogeny22Chain, respectively. We note that the failure analyses of RandomEquivalentPrimeIdeal and GeneralizedRepresentInteger follow closely the analysis in [AAA+25] with numbers adjusted to our setting. The analysis of Qlapoti follows that in [BCRSE+26]. Finally,

the analysis of Isogeny22Chain is almost identical to that in [AAA+25], hence we label that section with a star superscript.

9.1 Failure analysis of RandomEquivalentPrimeIdeal

The algorithm searches for an ideal equivalent to the input ideal I and of prime norm. In practice, this amounts to finding $\beta \in I$ such that $\mathrm{nrd}(\beta)/\mathrm{nrd}(I)$ is prime. Given a L2-reduced basis $\alpha_1, \alpha_2, \alpha_3, \alpha_4$ of I, this is equivalent to finding a prime represented by the quadratic form

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

Let $B := \texttt{QUAT\_equiv\_bound\_coeff}$, whose value is specified in Section 4.1.1. The algorithm samples c_1, c_2, c_3, c_4 from the box $[-B, B]$, yielding $(2B+1)^4$ candidates. By [DLRW24, Lemma 48], since $\alpha_1, \alpha_2, \alpha_3, \alpha_4$ form an L2-reduced basis, we have

$$q_I(c_1, c_2, c_3, c_4) \lesssim \frac{8p}{\pi^2} \cdot B^2.$$

Heuristic 9.1.1. Integers represented by a quadratic form behave like random integers of the same size in terms of primality and congruence conditions.

Following Heuristic 9.1.1 and treating the represented integers as random integers of this size, the probability that one candidate is prime is at least $1/(2\log(p))$. Consequently, the total failure probability is bounded by

$$\left(1 - \frac{1}{2\log(p)}\right)^{(2B+1)^4}.$$

We provide upper bounds on failure rates for our parameter choices in Table 9.1.

NGCC Level	p	QUAT_equiv_bound_coeff	failure rate
1	$69 \cdot 2^{313} - 1$	64	$< 2^{-904129}$
2	$27 \cdot 2^{500} - 1$	64	$< 2^{-571357}$
3	$15 \cdot 2^{1004} - 1$	64	$< 2^{-286030}$

Table 9.1: The upper bound of the failure rate of RandomEquivalentPrimeIdeal applied to QIMEN-PRISM parameters.

9.2 Failure analysis of GeneralizedRepresentInteger

GeneralizedRepresentInteger succeeds if the algorithm executes Line 9. Under Heuristic 9.1.1, the probability of M' being a prime congruent to 1 modulo 4 is approximately $1/(2\log M')$. Plugging in that in QIMEN-PRISM we choose $M = q(2^a - q)$ (as specified in Section 4.1) and that $M' := 4M - p(z^2 + t^2) > 0$, M' is then upper bounded by $p^{1.5}$ and consequently the probability of success is at least $1/(3\log(p))$.

Let B denote the number of iterations of the loop in GeneralizedRepresentInteger. The failure probability is then upper bounded by

$$(1 - \frac{1}{3\log(p)})^B.$$

As soon as B is bigger than $3\lambda_c \log p$, which is always the case in QIMEN-PRISM, the failure rate of GeneralizedRepresentInteger is upper bounded by a negligible rate $2^{-\lambda_c}$.

9.3 Failure analysis of Qlapoti

We follow the failure analysis of Qlapoti given in [BCRSE+26, Section 4]. Like in that work, we introduce the following random variables to model the failure probability of Qlapoti:

- let $\delta := \frac{n}{\sqrt{p}}$ be the random variable over the domain of all ideal classes $[I]$ with $I \subset \mathcal{O}_0$, where n denotes the norm of the smallest equivalent ideal in the class $[I]$;
- for fixed n, let $\epsilon_n := \frac{\sqrt{s^2+t^2}}{\sqrt{n}}$ denote the random variable over the domain $a, b, c \in \mathbb{Z}/n\mathbb{Z}$ with $\gcd(a, n) = 1$ and $n \nmid b$, and where (s, t) is a solution of smallest norm to the linear equation $ax + by = c \bmod n$.

For a discrete random variable X, we denote with f_X the probability mass function and with F_X the cumulative distribution function.

Following the analysis in [BCRSE+26, Section 4], for p of cryptographic size, we use δ_{wc} to denote the worst-case value of δ over all ideal classes $[I]$ with $I \subset \mathcal{O}_0$ and take $\delta_{wc} = 2\sqrt{2}/\pi$. Similarly, we use δ_{ac} to denote the average-case value of δ over all ideal classes $[I]$ with $I \subset \mathcal{O}_0$ and take $\delta_{ac} = 0.37$.

We calculate the number of α needed to find a *good* (α, s, t) tuple (meaning that z defined in Line 12 is positive) in the worst case and average case, denoted by $\#\alpha$ (wc) and $\#\alpha$ (ac), respectively. The number of α needed is estimated by first computing u_{wc} and u_{ac}, respectively, where $u := 1/\delta\sqrt{2f}$ with u_{wc} and u_{ac} computed by letting δ be δ_{wc} and δ_{ac}, respectively, and f is the cofactor in p. Following [BCRSE+26, Section 4], $\epsilon \leq 1/\delta\sqrt{2f}$. This means that the number of α's to try is explicitly given by $1/F_\epsilon(u)$. Since [BCRSE+26, Section 4] only contains a graph estimate of $F_\epsilon(u)$, we repeat their experiments and obtain the values of $\#\alpha$ as in Table 9.2.

NGCC Level	p	e	u_{wc}	$\#\alpha$ (wc)	u_{ac}	$\#\alpha$ (ac)
1	$69 \cdot 2^{313} - 1$	311	0.095	35.6	0.23	6.3
2	$27 \cdot 2^{500} - 1$	498	0.151	14.2	0.368	2.7
3	$15 \cdot 2^{1004} - 1$	1002	0.203	8.0	0.493	1.7

Table 9.2: Required worst-case, average-case numbers of α to generate a good tuple (α, s, t).

Now, taking into consideration that s, t, z needs to satisfy some congruence conditions, and that Line 23 needs to return a solution (for which we use the worst case analysis and require that z is a prime congruent to 1 modulo 4 in the analysis), this results in a worst case bound of

$$8 \log(\sqrt{p}/(2f\delta))$$

as indicated in [BCRSE+26, Equation (13)]. We obtain the new table on the required values for α for Qlapoti to succeed.

NGCC Level	p	e	$\#\alpha$ (wc)	$\#\alpha$ (ac)
1	$69 \cdot 2^{313} - 1$	311	30153	4510
2	$27 \cdot 2^{500} - 1$	498	19522	3715
3	$15 \cdot 2^{1004} - 1$	1002	22256	4734

Table 9.3: Required worst-case, average-case numbers of α for Qlapoti to finish.

According to the analysis of [BCRSE+26, Section 4], if n is the norm of the smallest ideal equivalent to the input ideal, then the expected number of generators α is

$$\approx 0.607927n \cdot \frac{2^{e-2}}{4p}.$$

Now we make the same assumption that the probability of success per α is the same for all α. The probability for Qlapoti to terminate therefore follows a geometric distribution. Let c denote the mean of this distribution, i.e. the average number of α's required to terminate, then the probability of success for a single α is $q = 1/c$. The probability that Qlapoti fails to terminate within k steps, i.e. k α's, is $(1-q)^k$, and we thus require

$$(1-q)^k < 2^{-(m+1)},$$

for some positive integer m. Taking logarithms and using $\log(1-x) \simeq -x$ for small x gives

$$k > (m+1)\log(2)c.$$

As a consequence, Qlapoti is guaranteed to succeed with failure probability less than $2^{-(m+1)}$ for any positive integer m if

$$n > (m+1) \cdot \frac{4\log(2) \cdot p \cdot c}{0.607927 \cdot 2^{e-2}}. \tag{9.1}$$

Here c denotes $\#\alpha$ (ac) from Table 9.3.

Following the conservative estimate of [BCRSE+26, Section 4], we analyze the failure probability of Qlapoti for QIMEN-PRISM parameters as follows: Let L denote the RHS of Eq. (9.1),

$$P(\text{Qlapoti fails}) < P(n \leq L) + P(\text{Qlapoti fails} \mid n > L) < F_\delta\left(\frac{L}{\sqrt{p}}\right) + 2^{-(m+1)}.$$

Note that here, we treat all instances such that $n \leq L$ as failures. For $\delta = n/\sqrt{p}$, the empirical estimate in [BCRSE+26, Section 4] gives that $F_\delta(x) \approx (6.46/2)x^2$. Hence,

$$P(\text{Qlapoti fails}) < F_\delta\left(\frac{L}{\sqrt{p}}\right) + 2^{-(m+1)} \approx \frac{6.46}{2}\left(\frac{L}{\sqrt{p}}\right)^2 + 2^{-(m+1)}.$$

Thus we obtain an upper bound on the failure rate by finding the largest m such that

$$2^{-(m+1)} > \frac{6.46}{2}(L/\sqrt{p})^2 = \frac{6.46}{2} \cdot \left((m+1) \cdot \frac{4\log(2) \cdot \sqrt{p} \cdot c}{0.607927 \cdot 2^{e-2}}\right)^2,$$

as this gives

$$P(\text{Qlapoti fails}) < 2^{-(m+1)} + 2^{-(m+1)} = 2^{-m}.$$

Specifically, for the QIMEN-PRISM parameter sets, we can read off c from Table 9.3, and obtain the final failure probabilities, which are confidently within the security levels (see Table 9.4).

NGCC Level	p	$e-2$	c	failure rate
1	$69 \cdot 2^{313} - 1$	311	4510	2^{-255}
2	$27 \cdot 2^{500} - 1$	498	3715	2^{-442}
3	$15 \cdot 2^{1004} - 1$	1002	4734	2^{-944}

Table 9.4: The upper bound of the failure rate of Qlapoti applied to the QIMEN-PRISM parameters.

Note that the first step of Qlapoti calls RandomEquivalentPrimeIdeal, the analysis above assumes this step succeeds and the failure rate can be taken into account easily by recalling the failure rate of RandomEquivalentPrimeIdeal calculated in Section 9.1.

9.4 Failure analysis of Isogeny22Chain*

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

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

The algorithms we gave in Section 2.5 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, where we nevertheless catch them and restart the process. If they happen during signing, they are propagated as signing failures. If they happen during verification, with overwhelming probability they indicate a malicious signature and thus lead to rejection.

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

Heuristic 9.4.1. The surfaces A_k encountered along the chains of $(2,2)$-isogenies computed in QIMEN-PRISM.KeyGen and QIMEN-PRISM.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 $\widetilde{O}(p^{-1})$.

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 ThetaChangeOfBasis 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 9.4.2. 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 QIMEN-PRISM.KeyGen, QIMEN-PRISM.Sign, and QIMEN-PRISM.Verif. Then the trace of the product coordinate $X_1 \cdot X_2$ under the gluing kernel behaves like an independent random element of $\mathbb{F}_{p^2}$.

Clearly the chance of encountering a zero trace is $O(p^{-2})$, and the computation only involves $O(1)$ two-dimensional gluing 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.

Bibliography

[AAA+25] 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. Technical report, National Institute of Standards and Technology, 2025.

[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. LNCS, pages 300–332. Springer, Cham, Switzerland, May 10–13, 2025.

[BBC+26] Andrea Basso, Giacomo Borin, Wouter Castryck, Maria Corte-Real Santos, Riccardo Invernizzi, Antonin Leroux, Luciano Maino, Frederik Vercauteren, and Benjamin Wesolowski. PRISM with a pinch of salt: Simple, efficient and strongly unforgeable signatures from isogenies. Cryptology ePrint Archive, Paper 2026/443, 2026.

[BBS+26] Andrea Basso, Giacomo Borin, Maria Corte-Real Santos, Pierrick Dartois, Riccardo Invernizzi, Luciano Maino, Robi Pedersen, and Michel Seck. Isogeny-based signatures with randomizable keys. Cryptology ePrint Archive, Paper 2026/1169, 2026.

[BCL08] Reinier Bröker, Denis Charles, and Kristin Lauter. Evaluating large degree isogenies and applications to pairing based cryptography. In Steven D. Galbraith and Kenneth G. Paterson, editors, *PAIRING 2008*, volume 5209 of *LNCS*, pages 100–112, Egham, UK, September 1–3, 2008. Springer, Berlin, Heidelberg, Germany.

[BCRSE+26] 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 Goichiro Hanaoka and Bo-Yin Yang, editors, *Advances in Cryptology – ASIACRYPT 2025*, pages 174–205, Singapore, 2026. Springer Nature Singapore.

[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*, LNCS, pages 339–370. Springer, Singapore, Singapore, December 7–11, 2024.

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

[BJS14] Jean-François Biasse, David Jao, and Anirudh Sankar. A quantum algorithm for computing isogenies between supersingular elliptic curves. In Willi Meier and Debdeep Mukhopadhyay, editors, *INDOCRYPT 2014*, volume 8885 of *LNCS*, pages 428–442, New Delhi, India, December 14–17, 2014. Springer, Cham, Switzerland.

[Can89] David G Cantor. On arithmetical algorithms over finite fields. *Journal of Combinatorial Theory, Series A*, 50(2):285–300, 1989.

[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*, pages 1696–1714, San Francisco, CA, USA, May 24–27, 2021. IEEE Computer Society Press.

[CEMR24] Maria Corte-Real Santos, Jonathan Komada Eriksen, Michael Meyer, and Krijn Reijnders. AprèsSQI: Extra fast verification for SQIsign using extension-field signing. In Marc Joye and Gregor Leander, editors, *EUROCRYPT 2024, Part I*, volume 14651 of *LNCS*, pages 63–93, Zurich, Switzerland, May 26–30, 2024. Springer, Cham, Switzerland.

[CGMT26] Jolijn Cottaar, Steven D. Galbraith, Luciano Maino, and Monika Trimoska. An analysis of a weakened version of PRISM. Cryptology ePrint Archive, Paper 2026/906, 2026.

[Coh93] Henri Cohen. *A Course in Computational Algebraic Number Theory*. Springer, New York, NY, USA, 1993.

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

[Cou96] Jean-Marc Couveignes. Computing ℓ-isogenies using the p-torsion. In *Algorithmic Number Theory*. Springer, 1996.

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

[DFH+24] Jelle Don, Serge Fehr, Yu-Hsuan Huang, Jyun-Jie Liao, and Patrick Struck. Hide-and-seek and the non-resignability of the BUFF transform. In *TCC 2024, Part III*, LNCS, pages 347–370. Springer, Cham, Switzerland, November 2024.

[DG16] Christina Delfs and Steven D. Galbraith. Computing isogenies between supersingular elliptic curves over $\mathbb{F}_p$. *DCC*, 78(2):425–440, 2016.

[DKL+20] Luca De Feo, David Kohel, Antonin Leroux, Christophe Petit, and Benjamin Wesolowski. SQISign: Compact post-quantum signatures from quaternions and isogenies. In Shiho Moriai and Huaxiong Wang, editors, *ASIACRYPT 2020, Part I*, volume 12491 of *LNCS*, pages 64–93, Daejeon, South Korea, December 7–11, 2020. Springer, Cham, Switzerland.

[DLLW23] Luca De Feo, Antonin Leroux, Patrick Longa, and Benjamin Wesolowski. New algorithms for the Deuring correspondence - towards practical and secure SQISign signatures. In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part V*, volume 14008 of *LNCS*, pages 659–690, Lyon, France, April 23–27, 2023. Springer, Cham, Switzerland.

[DLP93] Ivan Damgård, Peter Landrock, and Carl Pomerance. Average case error estimates for the strong probable prime test. *Mathematics of computation*, 61(203):177–194, 1993.

[DLRW24] Pierrick Dartois, Antonin Leroux, Damien Robert, and Benjamin Wesolowski. SQIsignHD: New dimensions in cryptography. In Marc Joye and Gregor Leander, editors, *EUROCRYPT 2024, Part I*, volume 14651 of *LNCS*, pages 3–32, Zurich, Switzerland, May 26–30, 2024. Springer, Cham, Switzerland.

[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*, LNCS, pages 304–338. Springer, Singapore, Singapore, December 7–11, 2024.

[Ebe07] David Eberly. The laplace expansion theorem: Computing the determinants and inverses of matrices. *Geometric Tools, LLC, Scottsdale, Ariz, USA*, 2007.

[Elk98] Noam D. Elkies. Elliptic and modular curves over finite fields and related computational issues. In *Computational Perspectives on Number Theory*. American Mathematical Society, 1998.

[FHK25] Serge Fehr, Yu-Hsuan Huang, and Julia Kastner. Sandwich BUFF: Achieving non-resignability using iterative hash functions. In *TCC 2025, Part III*, LNCS, pages 235–265. Springer, Cham, Switzerland, November 2025.

[FR94] Gerhard Frey and Hans-Georg Rück. A remark concerning m-divisibility and the discrete logarithm in the divisor class group of curves. *Mathematics of computation*, 62(206):865–874, 1994.

[Gro96] Lov K. Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*, STOC '96, page 212219, New York, NY, USA, 1996. Association for Computing Machinery.

[JAC+22] David Jao, Reza Azarderakhsh, Matthew Campagna, Craig Costello, Luca De Feo, Basil Hess, Amir Jalali, Brian Koziel, Brian LaMacchia, Patrick Longa, Michael Naehrig, Joost Renes, Vladimir Soukharev, David Urbanik, Geovandro Pereira, Koray Karabina, and Aaron Hutchinson. SIKE. Technical report, National Institute of Standards and Technology, 2022. available at https://csrc.nist.gov/Projects/post-quantum-cryptography/round-4-submissions.

[JD11] David Jao and Luca De Feo. Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies. In Bo-Yin Yang, editor, *Post-Quantum Cryptography - 4th International Workshop, PQCrypto 2011*, pages 19–34, Tapei, Taiwan, November 29 – December 2 2011. Springer, Berlin, Heidelberg, Germany.

[Koh96] David R. Kohel. *Endomorphism Rings of Elliptic Curves over Finite Fields*. PhD thesis, University of California, Berkeley, 1996.

[LDK+22] Vadim Lyubashevsky, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Peter Schwabe, Gregor Seiler, Damien Stehlé, and Shi Bai. CRYSTALS-DILITHIUM. Technical report, National Institute of Standards and Technology, 2022. available at https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022.

[Lic69] Stephen Lichtenbaum. Duality theorems for curves over p-adic fields. *Inventiones mathematicae*, 7(2):120–136, 1969.

[LLL82] A.K. Lenstra, H.W.jun. Lenstra, and László Lovász. Factoring polynomials with rational coefficients. *Math. Ann.*, 261:515–534, 1982.

[LM00] Reynald Lercier and François Morain. Computing isogenies between elliptic curves over F_{p^n} using couveigness algorithm. *Mathematics of Computation*, 69(229):351–370, 2000.

[LM26] Yi-Fu Lai and Luciano Maino. Toward zkSNARK-assisted isogeny-based cryptography. Cryptology ePrint Archive, Paper 2026/1096, 2026.

[LR23] David Lubicz and Damien Robert. Fast change of level and applications to isogenies. *Research in Number Theory*, 9(1):7, 2023.

[Mil86] James S Milne. Jacobian varieties. In *Arithmetic geometry*, pages 167–212. Springer, 1986.

[MN90] François Morain and Jean-Louis Nicolas. On Cornacchia's algorithm for solving the diophantine equation $u^2 + dv^2 = m$, 1990.

[Mon85] Peter L. Montgomery. Modular multiplication without trial division. *Mathematics of Computation*, 44(170):519–521, 1985.

[Mon87] Peter L. Montgomery. Speeding the Pollard and elliptic curve methods of factorization. *Mathematics of Computation*, 48(177):243–264, 1987.

[NO25] Kohei Nakagawa and Hiroshi Onuki. Attacks on PRISM-id via torsion over small extension fields. Cryptology ePrint Archive, Report 2025/1602, 2025.

[NOC+24] Kohei Nakagawa, Hiroshi Onuki, Wouter Castryck, Mingjie Chen, Riccardo Invernizzi, Gioella Lorenzon, and Frederik Vercauteren. SQIsign2D-East: A new signature scheme using 2-dimensional isogenies. In *ASIACRYPT 2024, Part III*, LNCS, pages 272–303. Springer, Singapore, Singapore, December 7–11, 2024.

[NS09] Phong Q. Nguyen and Damien Stehlé. An lll algorithm with quadratic complexity. *SIAM Journal on Computing*, 39(3):874–903, 2009.

[NV10] Phong Q Nguyen and Brigitte Vallée. *The LLL algorithm*. Springer, 2010.

[PFH+22] Thomas Prest, Pierre-Alain Fouque, Jeffrey Hoffstein, Paul Kirchner, Vadim Lyubashevsky, Thomas Pornin, Thomas Ricosset, Gregor Seiler, William Whyte, and Zhenfei Zhang. FALCON. Technical report, National Institute of Standards and Technology, 2022. available at https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022.

[PH78] Stephen Pohlig and Martin Hellman. An improved algorithm for computing logarithms over gf(p) and its cryptographic significance. *IEEE Transactions on Information Theory*, 24(1):106–110, January 1978.

[PRR+25] Giacomo Pope, Krijn Reijnders, Damien Robert, Alessandro Sferlazza, and Benjamin Smith. Simpler and faster pairings from the montgomery ladder. *CiC*, 2(2):29, 2025.

[PZ24] Patrick Pelissier and Paul Zimmerman. The DPE library, 2024.

[RS17] Joost Renes and Benjamin Smith. qDSA: Small and secure digital signatures with curve-based Diffie-Hellman key pairs. In Tsuyoshi Takagi and Thomas Peyrin, editors, *ASIACRYPT 2017, Part II*, volume 10625 of *LNCS*, pages 273–302, Hong Kong, China, December 3–7, 2017. Springer, Cham, Switzerland.

[Sco24] Michael Scott. Modular arithmetic for cryptographers. Technology Innovation Institute, White Paper, 2024.

[Tat62] John Tate. Duality theorems in galois cohomology over number fields. In *Proc. Internat. Congr. Mathematicians (Stockholm, 1962)*, pages 288–295, 1962.

[Thr] Prism: Compact threshold signatures from pushforwards of large-degree isogenies. https://csrc.nist.gov/presentations/2026/mpts2026-4a2. Accessed: 2026-06-17.

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

[Wei40] André Weil. Sur les fonctions algebriques á corps de constantes finis, volume i. *Oeuvres Scientifiques, Paris*, pages 257–259, 1940.

[Wei57] André Weil. *Zum Beweis des Torellischen Satzes: CL Siegel zum 60. Geburtstage*. Vandenhoeck & Ruprecht, 1957.

[Wes22] Benjamin Wesolowski. The supersingular isogeny path and endomorphism ring problems are equivalent. In *62nd FOCS*, pages 1100–1111, Denver, CO, USA, February 7–10, 2022. IEEE Computer Society Press.

[ZSP+18] Gustavo Zanon, Marcos A. Simplício, Jr., Geovandro C. C. F. Pereira, Javad Doliskani, and Paulo S. L. M. Barreto. Faster isogeny-based compressed key agreement. In Tanja Lange and Rainer Steinwandt, editors, *Post-Quantum Cryptography - 9th International Conference, PQCrypto 2018*, pages 248–268, Fort Lauderdale, Florida, United States, April 9–11, 2018. Springer, Cham, Switzerland.

CHAPTER A
Overview of Implementation

In this brief chapter, we provide an overview of the different modules required for an implementation of QIMEN-PRISM, together with how these modules interact. The other chapters of the appendix provide the algorithms within these modules.

`intbig`: provides multiprecision arithmetic required for quaternion arithmetic (`quat`).

`gf`: provides finite field arithmetic required for elliptic curve arithmetic (`ec`). We give a full description of this module in Chapter B.

`ec`: provides the core elliptic curve arithmetic, in particular core algorithms for higher-dimensional isogenies (`hd`) and ideal translation (`qlapoti`). We give a full description of this module in Chapter C.

`hd`: provides the necessary algorithms to compute chains of $(2, 2)$-isogenies, which are used in ideal translation (`qlapoti`) and the core scheme functionalities (QIMEN-PRISM.KeyGen, QIMEN-PRISM.Sign, QIMEN-PRISM.Verif). We give a full description of this module in Chapter D.

`biext`: provides the cubical arithmetic required to compute Weil and Tate pairings. In the implementation, this is a submodule of `ec`. However, due to the complexity of this submodule, we describe it separately in Chapter E.

`quat`: provides the core quaternion arithmetic, in particular core algorithms for ideal translation (`qlapoti`). We give a full description of this module in Chapter F.

`qlapoti`: provides the algorithms to translate ideals to isogenies, and vice versa, as used in `hd` and the core scheme functionalities (QIMEN-PRISM.KeyGen, QIMEN-PRISM.Sign, QIMEN-PRISM.Verif).

There are also two additional modules: `precomp` provides necessary precomputation of several constants or scheme parameters as used in several modules, and `common` provides basis cryptographic functionalities, such as hash functions and PRNGs.

Figure A.1 is a visualisation of the interdependence of these modules, together with the key protocol algorithms QIMEN-PRISM.KeyGen, QIMEN-PRISM.Sign, QIMEN-PRISM.Verif.

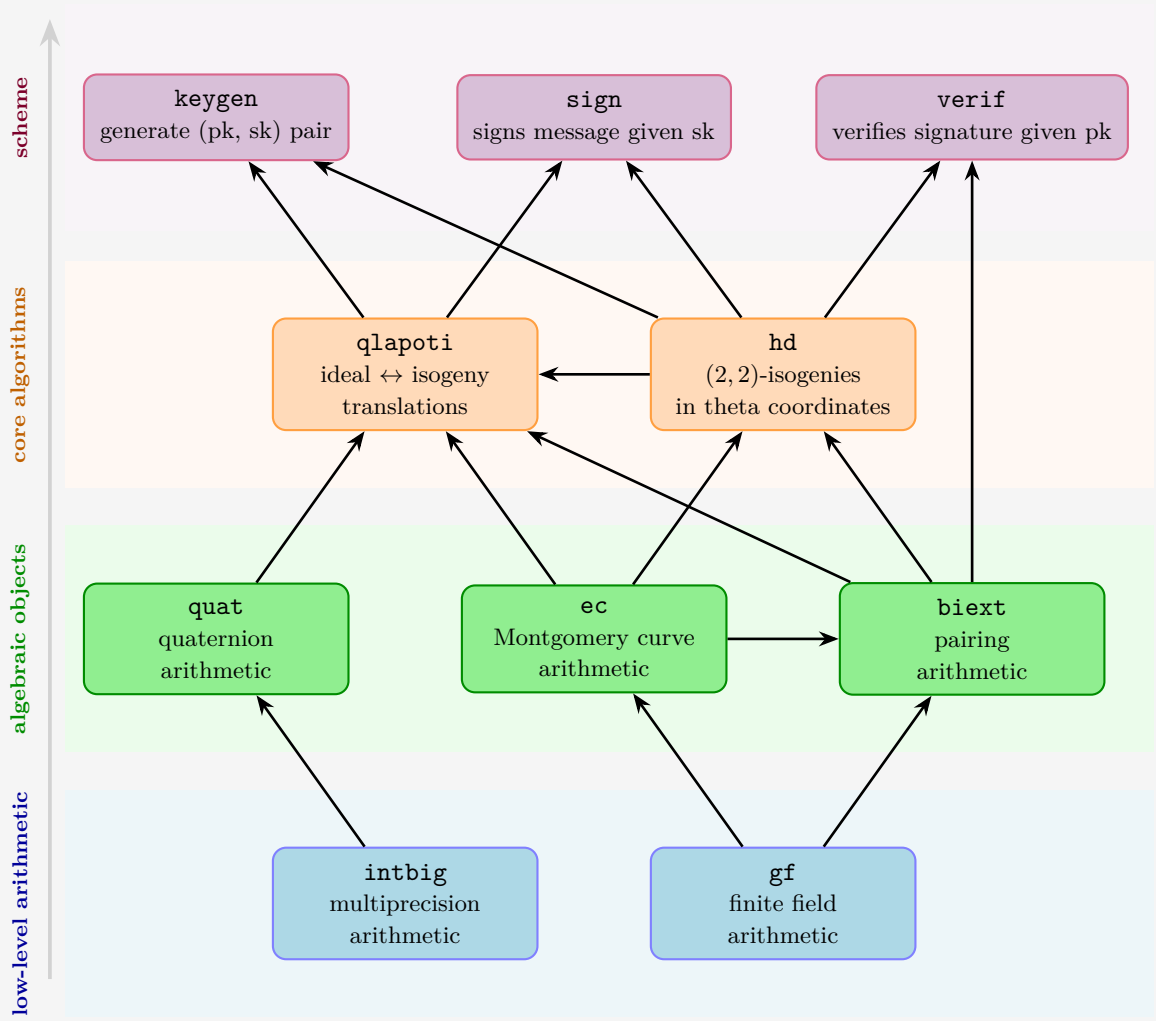

Figure A.1: Module structure of the reference implementation of QIMEN-PRISM, going up in abstraction vertically.

CHAPTER B

Finite field arithmetic (implementation details)*

All higher-level operations in QIMEN-PRISM reduce to arithmetic in $\mathbb{F}_p$ and its quadratic extension $\mathbb{F}_{p^2} = \mathbb{F}_p(i)$ (Section 2.1). The implementation has three layers:

1. an $\mathbb{F}_p$ Montgomery arithmetic kernel, auto-generated by Scott's code generator [Sco24] in unsaturated-radix form;
2. an encoding/decoding wrapper with constant-time utilities; and
3. $\mathbb{F}_{p^2}$ arithmetic built on top of $\mathbb{F}_p$.

All code is portable C99, using `__uint128_t` for double-width products. It targets 64-bit platforms and runs in constant time on all secret-dependent inputs.

B.1 Element representation

Each element of $\mathbb{F}_p$ is stored as an array of n unsigned 64-bit limbs in *unsaturated radix* representation: each limb carries at most $r < 64$ significant bits. An element $a \in \mathbb{F}_p$ is encoded as $(a_0, a_1, \ldots, a_{n-1})$ with

$$a \equiv \sum_{j=0}^{n-1} a_j \cdot 2^{r\,j} \pmod{p}, \qquad 0 \le a_j < 2^r.$$

The $64 - r$ slack bits per limb absorb intermediate carries, so additions and subtractions proceed without immediate carry propagation.

Internally, all $\mathbb{F}_p$ elements are held in Montgomery form [Mon85]: the stored value for a is $\tilde{a} = a \cdot R \bmod p$ with $R = 2^{n \cdot r}$. Conversion (`nres`/`redc`) occurs only at import/export boundaries. An element of $\mathbb{F}_{p^2} = \mathbb{F}_p(i)$ is a pair $(\mathsf{re}, \mathsf{im})$ of $\mathbb{F}_p$ elements representing $\mathsf{re} + \mathsf{im} \cdot i$.

For serialization, an $\mathbb{F}_p$ element is reduced via `redc` and written as $\ell_p = \lceil \log_2 p/8 \rceil$ bytes in little-endian order. An $\mathbb{F}_{p^2}$ element is the concatenation of its real and imaginary parts, occupying $2\ell_p$ bytes.

B.2 Arithmetic in $\mathbb{F}_p$

The $\mathbb{F}_p$ arithmetic kernel is generated by Scott's `monty.py` tool [Sco24], which produces portable C code for the unsaturated radix-2^r Montgomery representation described in Section B.1. In this setting, the unsaturated layout is used to keep carry handling simple in a high-level implementation: carries are propagated by shifts and masks, and the reduction logic is expressed directly at the limb level. The generated code is further checked by the script against random test inputs, together with additional overflow checks. The kernel also includes constant-time conditional selection and swap implemented by bitwise masking, without secret-dependent branches or memory-access patterns.

- **Modular addition and subtraction** are implemented at the limb level. Addition consists of a limb-wise sum followed by a carry-propagation pass (`prop`), which transfers excess bits from each limb to the next in $O(n)$ word operations. When the intermediate result is negative, a branch-free correction step (`flatten`) adds back p. Subtraction is handled analogously. Throughout the arithmetic, intermediate values are generally maintained below $2p$ rather than reduced to the canonical interval after every operation.
- **Multiplication** merges the schoolbook product with Montgomery reduction in a single pass. For each column, the relevant partial sum is accumulated, the corresponding reduction digit is determined, and the contribution of that digit is folded into the running state immediately, so that no full double-width intermediate is materialized. Each limb product is obtained via the `__uint128_t` intrinsic.
- **Squaring** uses the symmetry $a_j a_k = a_k a_j$ to reduce the number of off-diagonal products.
- **Inversion**, **square roots**, and **Legendre symbols** are expressed through a common addition-chain exponentiation, called the progenitor in [Sco24], which computes $\pi = a^{(p-3)/4}$. Since $p \equiv 3 \pmod 4$, the square root is recovered as $\sqrt{a} = \pi \cdot a = a^{(p+1)/4}$; see Equation (2.1). The Legendre symbol is obtained as $\pi^2 \cdot a = a^{(p-1)/2}$, and the inverse as $\pi^4 \cdot a = a^{p-2}$.

B.3 Arithmetic in $\mathbb{F}_{p^2}$

Most arithmetic in $\mathbb{F}_{p^2}$ is reduced to $\mathbb{F}_p$ arithmetic.

- We perform addition, subtraction, negation, and halving in $\mathbb{F}_{p^2}$ component-wise, costing costing two $\mathbb{F}_p$ operations each,
- We use Algorithm 22 for multiplication in $\mathbb{F}_{p^2}$, which gives a Karatsuba-like formula that reduces the cost from four $\mathbb{F}_p$ multiplications (Section 2.1.2) to three, at the expense of two extra additions.
- We use Algorithm 23 for squaring in $\mathbb{F}_{p^2}$, which uses the identity $(a_0+a_1 i)^2 = (a_0+a_1)(a_0-a_1)+2a_0a_1\, i$ hence requires only two $\mathbb{F}_p$ multiplications,
- inversion uses the norm-based method of Section 2.1.2: compute $N = a_0^2 + a_1^2 \in \mathbb{F}_p$, invert N, and scale the conjugate $(a_0, -a_1)$ by N^{-1},
- batch inversion on k elements uses Montgomery's batched-inversion trick [Mon87], reducing k inversions to one inversion plus $3(k-1)$ $\mathbb{F}_{p^2}$ multiplications: prefix products are accumulated forward, the total is inverted, and individual inverses are recovered in reverse.
- a quadratic reciprocity check for $a \in \mathbb{F}_{p^2}$ is reduced to checking that $a_0^2 + a_1^2$ is a square in $\mathbb{F}_p$ using Section 2.1.2,
- Square roots are computed via Eq. (2.2) using the $\mathbb{F}_p$ progenitor; the two cases ($\chi = 1$ vs. $\chi = -1$) are resolved by constant-time conditional selection.

We give a summary of these costs in Table B.1 for use in subsequent sections.

Table B.1: Cost notation used throughout the implementation chapter.

Symbol	Operation	$\mathbb{F}_p$-cost
M	one $\mathbb{F}_{p^2}$ multiplication	3 $\mathbb{F}_p$ mul and 6 $\mathbb{F}_p$ adds
S	one $\mathbb{F}_{p^2}$ squaring	2 $\mathbb{F}_p$ mul and 3 $\mathbb{F}_p$ adds
a	one $\mathbb{F}_{p^2}$ addition or subtraction	2 $\mathbb{F}_p$ add/subs

Algorithm 22 Fp2Mul(a, b)

Input: $a = a_0 + a_1 i,\ b = b_0 + b_1 i \in \mathbb{F}_{p^2}$
Output: $c = a \cdot b \in \mathbb{F}_{p^2}$

$t_0 \leftarrow (a_0 + a_1) \cdot (b_0 + b_1)$
$t_1 \leftarrow a_1 \cdot b_1$
$c_0 \leftarrow a_0 \cdot b_0$
$c_1 \leftarrow t_0 - t_1 - c_0$ ▷ $= a_0 b_1 + a_1 b_0$
$c_0 \leftarrow c_0 - t_1$ ▷ $= a_0 b_0 - a_1 b_1$
return $c = c_0 + c_1\, i$

Algorithm 23 Fp2Sqr(a)

Input: $a = a_0 + a_1 i \in \mathbb{F}_{p^2}$
Output: $c = a^2 \in \mathbb{F}_{p^2}$

$c_1 \leftarrow 2\, a_0 \cdot a_1$
$c_0 \leftarrow (a_0 + a_1) \cdot (a_0 - a_1)$ ▷ $= a_0^2 - a_1^2$
return $c = c_0 + c_1\, i$

B.4 Discrete logarithms

In this section we describe the finite-field discrete logarithm routines. These computations take place in multiplicative subgroups of $\mathbb{F}_{p^2}^{\times}$. QIMEN-PRISM uses only the dyadic case. Let $\zeta \in \mu_{2^e} \subset \mathbb{F}_{p^2}^{\times}$ be an element of order 2^e, and let $\eta = \zeta^k$. The standard recursive method recovers the least significant bit of k from $\eta^{2^{e-1}} \in \{\pm 1\}$, removes this contribution, and then reduces the problem from order 2^e to order 2^{e-1}. Repeating this step recovers the full binary expansion of k.

Algorithm 24 DLogPowerOfTwo(ζ, η, e)

Input: $\zeta \in \mathbb{F}_{p^2}^{\times}$ of order 2^e, $\eta = \zeta^k \in \langle \zeta \rangle$
Output: $k \in \mathbb{Z}/2^e\mathbb{Z}$ such that $\eta = \zeta^k$

if $e = 1$ **then**
    if $\eta = 1$ **then**
        return 0
    else
        return 1
if $\eta^{2^{e-1}} = 1$ **then**
    $b \leftarrow 0$
else
    $b \leftarrow 1$
$\eta' \leftarrow \eta \cdot \zeta^{-b}$
$k' \leftarrow$ DLogPowerOfTwo$(\zeta^2, \eta', e - 1)$
return $b + 2k' \bmod 2^e$

CHAPTER C

Elliptic curve arithmetic (implementation details)*

Recall from Section 2.2.1 that we always work with Montgomery curves over $\mathbb{F}_{p^2}$. The curve coefficient is stored in projective form as a pair $(A : C)$, such that A/C is the Montgomery coefficient, together with the precomputed quantity $(A_{24} : C_{24}) = (A + 2C : 4C)$ to accelerate point doublings.

C.1 Projective x-only arithmetic

Points are represented in projective x-only coordinates $(X : Z)$ such that $x = X/Z$. Except where explicitly required, the y-coordinate is omitted, since the isogeny procedures operate only on x-coordinate data. Under this representation, a point is determined only up to sign, which is sufficient for the procedures specified below. Two fundamental algorithms are xDBL and xADD. These algorithms provide the constant-time primitives for the scalar multiplication used in the protocol.

- xDBL (Algorithm 25) computes the doubling of a point using the projective curve constants $(A_{24} : C_{24})$ at a cost of $2\mathrm{S} + 4\mathrm{M} + 4\mathrm{a}$,
- xADD (Algorithm 26) computes $P + Q$ from P, Q, and the known difference $P - Q$ at a cost of $2\mathrm{S} + 4\mathrm{M} + 6\mathrm{a}$,
- xDBLADD (Algorithm 27) combines xDBL and xADD in a single routine and reuses intermediate values, resulting in a total cost of $4\mathrm{S} + 8\mathrm{M} + 8\mathrm{a}$,
- Ladder computes a scalar multiple $[m]P$, given m and P. When m is a k-bit scalar, this costs one call to xDBLADD per bit, for a total of $4k\mathrm{S} + 8k\mathrm{M} + 8k\mathrm{a}$.
- Ladder3pt computes $P + [m]Q$, given m, P, Q and their difference, and a similar cost as Ladder.
- LadderBiscalar computes $[m]P + [n]Q$, given m, n, P, Q and their difference. When m and n are both k bits, this starts with one call to xADD, and each loop iteration then performs one call to xDBL and two calls to xADD for a total cost of $(6k + 2)\mathrm{S} + (12k + 4)\mathrm{M} + (16k + 6)\mathrm{a}$.

Algorithm 25 xDBL$(P, (A_{24} : C_{24}))$

Require: $P = (X_P : Z_P)$, Montgomery constants $(A_{24} : C_{24})$ of curve E
Ensure: $[2]P = (X_{2P} : Z_{2P})$

1. $t_0 \leftarrow X_P + Z_P$
2. $t_1 \leftarrow X_P - Z_P$
3. $t_0 \leftarrow t_0^2$
4. $t_1 \leftarrow t_1^2$
5. $t_2 \leftarrow t_0 - t_1$
6. $Z_{2P} \leftarrow t_1 \cdot C_{24}$
7. $X_{2P} \leftarrow t_0 \cdot Z_{2P}$
8. $t_0 \leftarrow t_2 \cdot A_{24}$
9. $t_0 \leftarrow t_0 + Z_{2P}$
10. $Z_{2P} \leftarrow t_0 \cdot t_2$
11. **return** $(X_{2P} : Z_{2P})$

▷ Cost: $2\mathrm{S} + 4\mathrm{M} + 4\mathrm{a}$

Algorithm 26 xADD$(P, Q, P-Q)$

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

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

Algorithm 27 xDBLADD$(P, Q, P-Q, (A_{24} : C_{24}))$

Require: $P = (X_P : Z_P)$, $Q = (X_Q : Z_Q)$, $P - Q = (X_{P-Q} : Z_{P-Q})$, Montgomery constants $(A_{24} : C_{24})$ of E
Ensure: $[2]P = (X_{2P} : Z_{2P})$ and $P + Q = (X_{P+Q} : Z_{P+Q})$

1. $t_0 \leftarrow X_P + Z_P$
2. $t_1 \leftarrow X_P - Z_P$
3. $X_{2P} \leftarrow t_0^2$
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** $\big((X_{2P} : Z_{2P}),\ (X_{P+Q} : Z_{P+Q})\big)$ ▷ Cost: $4\mathrm{S} + 8\mathrm{M} + 8\mathrm{a}$

Algorithm 28 Ladder(P, E, m)

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

$((X_0 : Z_0), (X_1 : Z_1)) \leftarrow ((1 : 0), (X_P : Z_P))$
for $i \leftarrow k - 1$ **down to** 0 **do**
  if $m_i = 1$ **then**
    $((X_1 : Z_1), (X_0 : Z_0)) \leftarrow$ xDBLADD$((X_1 : Z_1), (X_0 : Z_0), (X_P : Z_P), (A_{24} : C_{24}))$
  else
    $((X_0 : Z_0), (X_1 : Z_1)) \leftarrow$ xDBLADD$((X_0 : Z_0), (X_1 : Z_1), (X_P : Z_P), (A_{24} : C_{24}))$
$(X_{[m]P} : Z_{[m]P}) \leftarrow (X_0 : Z_0)$
return $[m]P = (X_{[m]P} : Z_{[m]P})$ ▷ Cost: $4k\mathrm{S} + 8k\mathrm{M} + 8k\mathrm{a}$

Algorithm 29 Ladder3pt$(P, Q, P-Q, (A_{24} : C_{24}), m)$

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

$((X_0 : Z_0), (X_1 : Z_1), (X_2 : Z_2)) \leftarrow ((X_P : Z_P), (X_Q : Z_Q), (X_{P-Q} : Z_{P-Q}))$
for $i \leftarrow 0$ **to** $k - 1$ **do**
  if $m_i = 1$ **then**
    $((X_0 : Z_0), (X_1 : Z_1)) \leftarrow$ xDBLADD$((X_0 : Z_0), (X_1 : Z_1), (X_2 : Z_2), (A_{24} : C_{24}))$
  else
    $((X_0 : Z_0), (X_2 : Z_2)) \leftarrow$ xDBLADD$((X_0 : Z_0), (X_2 : Z_2), (X_1 : Z_1), (A_{24} : C_{24}))$
$(X_{P+[m]Q} : Z_{P+[m]Q}) \leftarrow (X_1 : Z_1)$
return $P + [m]Q = (X_{P+[m]Q} : Z_{P+[m]Q})$ ▷ Cost: $4k\mathrm{S} + 8k\mathrm{M} + 8k\mathrm{a}$

Algorithm 30 LadderBiscalar$(P, Q, P-Q, (A_{24} : C_{24}), m, n)$

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

$(\sigma_0, \sigma_1) \leftarrow (1, 0)$ if m is even and n is odd; otherwise $(\sigma_0, \sigma_1) \leftarrow (0, 1)$
$m' \leftarrow m,\ n' \leftarrow n$
if m is even **then**
    $m' \leftarrow m' - 1$
if n is even **then**
    $n' \leftarrow n' - 1$
$b_0 \leftarrow (0m'_{k-1} \cdots m'_0)_2,\ b_1 \leftarrow (0n'_{k-1} \cdots n'_0)_2,\ b \leftarrow (b_0, b_1)$
for $i \leftarrow 0$ **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**
        $(\sigma_0, \sigma_1) \leftarrow (\sigma_1, \sigma_0)$
$R_0 \leftarrow (1 : 0),\ T = (T_0, T_1) \leftarrow (P, Q),\ R_1 \leftarrow T_{\sigma_0},\ R_2 \leftarrow T_{(\sigma_0+1) \bmod 2}$
$D_1 \leftarrow R_1,\ D_2 \leftarrow R_2,\ R_2 \leftarrow \text{xADD}(R_1, R_2, P - Q)$
$F_1 \leftarrow R_2,\ F_2 \leftarrow P - Q$
for $i \leftarrow k - 1$ **down to** 0 **do**
    $h \leftarrow r_{2i} + r_{2i+1},\ T_0 \leftarrow R_{h \bmod 2},\ T \leftarrow (T_0, R_2)$
    $T_0 \leftarrow \text{xDBL}\big(T_{\lfloor h/2 \rfloor}, (A_{24} : C_{24})\big),\ T_1 \leftarrow R_{r_{2i+1}},\ T_2 \leftarrow R_{r_{2i+1}+1}$
    if $r_{2i+1} = 1$ **then**
        $(D_1, D_2) \leftarrow (D_2, D_1)$
    $T_1 \leftarrow \text{xADD}(T_1, T_2, D_1),\ T_2 \leftarrow \text{xADD}(R_0, R_2, F_1)$
    if $h \bmod 2 = 1$ **then**
        $(F_1, F_2) \leftarrow (F_2, F_1)$
    $(R_0, R_1, R_2) \leftarrow (T_0, T_1, T_2)$
$(X_{[m]P+[n]Q} : Z_{[m]P+[n]Q}) \leftarrow R_{((m \bmod 2)\oplus 1)+((n \bmod 2)\oplus 1)}$
return $[m]P + [n]Q = (X_{[m]P+[n]Q} : Z_{[m]P+[n]Q})$ ▷ $(6k + 2)$S + $(12k + 4)$M + $(16k + 6)$a

C.2 Projective x-only auxiliary routines

We use two auxiliary x-only routines.

- IsomorphismMontgomeryCurves pushes points through an isomorphism between Montgomery curves, following Section 2.2.1.
- ProjectiveDifference computes a deterministic choice of $x(P \pm Q)$ from $x(P)$ and $x(Q)$, following the description of Section 2.2.2.3.
- RecoverCodomain computes the curve coefficient $(A : C)$, given two points P, Q and their difference $P - Q$ in x-only projective form.

Algorithm 31 IsomorphismMontgomeryCurves(E, P, Q, E')

Input: Montgomery coefficients $(A : C)$ and $(A' : C')$ of the curves E and E', respectively, and points $P = (X_P : Z_P)$, $Q = (X_Q : Z_Q)$ on E
Output: The images $P' = (X_{P'} : Z_{P'})$, $Q' = (X_{Q'} : Z_{Q'})$ of P and Q under an isomorphism between E and E'

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

Algorithm 32 ProjectiveDifference$(P, Q, (A : C))$

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

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

Algorithm 33 RecoverCodomain$(P, Q, P - Q)$

Require: $P_1 = P = (X_1 : Z_1)$, $P_2 = Q = (X_2 : Z_2)$, and $P_3 = P - Q = (X_3 : Z_3)$
Ensure: Montgomery curve coefficient $(A : C)$

1. $x_{123} \leftarrow X_1X_2X_3$
2. $z_{123} \leftarrow Z_1Z_2Z_3$
3. $xz_1 \leftarrow X_1Z_2Z_3$
4. $xz_2 \leftarrow X_2Z_1Z_3$
5. $xz_3 \leftarrow X_3Z_1Z_2$
6. $t_1 \leftarrow xz_1 + xz_2 + xz_3$
7. $t_2 \leftarrow (X_1 - Z_1)(X_2 - Z_2)(X_3 - Z_3)$
8. $A \leftarrow t_2 - x_{123} - t_1 + 2z_{123}$
9. $x_{123} \leftarrow 4x_{123}$
10. $C \leftarrow x_{123}z_{123}$
11. $A \leftarrow A^2 - x_{123}t_1$
12. **return** $(A : C)$

▷ Cost: 1S + 14M + 12a

C.3 Jacobian Coordinates

Several steps of the QIMEN-PRISM protocol require the y-coordinate of a point, which x-only arithmetic ignores. For instance, after evaluating a $(2, 2)$-isogeny chain the resulting image points must be expressed as linear combinations of a torsion basis via discrete logarithms on small subgroups. Determining the correct sign of each point requires knowing y. The same is true during key generation, where the endomorphism θ is evaluated at non-smooth torsion points through a double-scalar multiplication $\theta(P) = [a]P + [b]\iota(P)$ that is sign-sensitive, and during decryption, where the torsion basis on E_B must be lifted to full coordinates in order to orient the kernel of the $(2, 2)$-isogeny correctly. In all these situations we switch to Jacobian coordinates $(X : Y : Z)$ satisfying $BY^2 = X^3 + AX^2Z^2 + XZ^4$. The conversion from Montgomery to

Jacobian form and back is given by

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

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

- DBL (Algorithm 35) takes as input the Jacobian coordinates of a point P and the normalized Montgomery coefficient $a = A/C$, and outputs the Jacobian coordinates of the doubled point $[2]P$.
- ADD (Algorithm 34) takes as input the Jacobian coordinates of two points P and Q, together with the normalized Montgomery coefficient $a = A/C$, and outputs the Jacobian coordinates of the sum $P + Q$.
- ADDComponents (Algorithm 36) takes as input the Jacobian coordinates of two distinct points P, Q and the normalized Montgomery coefficient $a = A/C$, and outputs (u, v, w) such that the projective x-only coordinates of $P + Q$ and $P - Q$ are given by

$$x(P + Q) = (u - v : w), \qquad x(P - Q) = (u + v : w).$$

Algorithm 34 ADD$(P, Q, (A : C))$

Require: Jacobian points $P = (X_P : Y_P : Z_P)$ and $Q = (X_Q : Y_Q : Z_Q)$ on the Montgomery curve E, with $P \neq Q$, $Q \neq -P$, and $P, Q \neq \mathcal{O}$
Ensure: Jacobian point $P + Q = (X_{P+Q} : Y_{P+Q} : Z_{P+Q})$

1. $t_0 \leftarrow Z_P^2$
2. $t_1 \leftarrow t_0 \cdot Z_P$
3. $t_2 \leftarrow Z_Q^2$
4. $t_3 \leftarrow t_2 \cdot Z_Q$
5. $t_1 \leftarrow t_1 \cdot Y_Q$
6. $t_3 \leftarrow t_3 \cdot Y_P$
7. $t_1 \leftarrow t_1 - t_3$
8. $t_0 \leftarrow t_0 \cdot X_Q$
9. $t_2 \leftarrow t_2 \cdot X_P$
10. $t_4 \leftarrow t_0 - t_2$
11. $t_0 \leftarrow t_0 + t_2$
12. $t_5 \leftarrow Z_P \cdot Z_Q$
13. $Z_{P+Q} \leftarrow t_4 \cdot t_5$
14. $t_5 \leftarrow t_5^2$
15. $t_5 \leftarrow t_5 \cdot A$
16. $t_0 \leftarrow t_0 + t_5$
17. $t_6 \leftarrow t_4^2$
18. $t_5 \leftarrow t_0 \cdot t_6$
19. $X_{P+Q} \leftarrow t_1^2$
20. $X_{P+Q} \leftarrow X_{P+Q} - t_5$
21. $t_3 \leftarrow t_3 \cdot t_4$
22. $t_3 \leftarrow t_3 \cdot t_6$
23. $t_2 \leftarrow t_2 \cdot t_6$
24. $Y_{P+Q} \leftarrow t_2 - X_{P+Q}$
25. $Y_{P+Q} \leftarrow Y_{P+Q} \cdot t_1$
26. $Y_{P+Q} \leftarrow Y_{P+Q} - t_3$
27. **return** $(X_{P+Q} : Y_{P+Q} : Z_{P+Q})$ ▷ Cost: 5S + 15M + 7a

Algorithm 35 DBL$(P, (A : C))$

Require: Jacobian point $P = (X_P : Y_P : Z_P)$ and Montgomery coefficient $(A : C)$ of curve E
Ensure: Jacobian point $[2]P = (X_{[2]P} : Y_{[2]P} : Z_{[2]P})$

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

Algorithm 36 ADDCOMPONENTS$(P, Q, (A : C))$

Require: Distinct Jacobian points $P = (X_P : Y_P : Z_P)$, $Q = (X_Q : Y_Q : Z_Q)$ and Montgomery coefficient $(A : C)$ of curve E
Ensure: (u, v, w) such that the x-only coordinates of $P + Q$ and $P - Q$ are $P + Q = (u - v : w)$ and $P - Q = (u + v : w)$

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

▷ Cost: $5\mathrm{S} + 11\mathrm{M} + 7\mathrm{a}$

CHAPTER D

2-Dimensional isogenies (implementation details)*

We now describe the implementation of two-dimensional isogenies of degree 2^n, namely $(2^n, 2^n)$-isogenies, in level-2 theta coordinates. The description of the $(2, 2)$-isogeny chain given below is adapted from the algorithmic treatment of Dartois, Maino, Pope, and Robert [DMPR24]. Throughout, operation counts are given in terms of squarings, multiplications, inversions, and additions over the base field $\mathbb{F}_q$, denoted by S, M, I, and a, respectively. We first describe the arithmetic of theta coordinates, which are used in the computation of such isogeny chains.

- Section D.1 introduces theta coordinates of level 2, and
- Section D.2 describes the formulas for doubling a point using theta coordinates,

At a high level, a $(2, 2)$-isogeny chain $E_1 \times E_2 \to E_3 \times E_4$ between two products of elliptic curves is handled in three stages.

1. We perform a gluing step from a product surface $E_1 \times E_2$ to a principally polarized abelian surface equipped with a fixed theta structure, see Section D.4,
2. We iterate generic $(2, 2)$-isogenies entirely in theta coordinates, see Section D.3,
3. After the last codomain has become isomorphic to a product again, we compute an explicit splitting isomorphism and return to product coordinates, see Section D.5.

D.1 Theta coordinates of level 2

For arithmetic on principally polarized abelian surfaces, we work with level-2 theta coordinates rather than Jacobian coordinates. In the same way that projective x-only coordinates provide an efficient model for Montgomery curves when the sign of the point is irrelevant, level-2 theta coordinates provide a projective model that is well adapted to $(2, 2)$-isogeny formulas. In particular, they support efficient doubling, codomain recovery, and isogeny evaluation in a uniform framework.

D.1.1 On Montgomery curves

Let E be a Montgomery curve over $\mathbb{F}_{p^2}$ defined by

$$By^2 = x^3 + Ax^2 + x, \qquad A, B \in \mathbb{F}_{p^2}.$$

We represent points on E in projective x-only coordinates $(X : Z)$. Level-2 theta coordinates give another projective model for the same geometric points, still defined only up to sign. Fix a basis (T_1', T_2') of $E[4]$ such that

$$T_1' = (-1 : 1), \qquad T_2' = (r : s),$$

The associated theta null point is

$$(a : b) = (r + s : r - s).$$

Once $(a : b)$ is fixed, change of coordinates between Montgomery and theta coordinates is given by

$$(X : Z) \longmapsto (\theta_0 : \theta_1) = \big(a(X - Z) : b(X + Z)\big),$$

and conversely by

$$(\theta_0 : \theta_1) \longmapsto (X : Z) = \big(a\theta_1 + b\theta_0 : a\theta_1 - b\theta_0\big),$$

We use the convention that the neutral point is represented by $(1 : 0)$ in Montgomery coordinates. In practice, this conversion is used whenever a point enters or leaves the two-dimensional part of the computation, and it is summarized in Algorithm 37.

Algorithm 37 MontgomeryToTheta$(P, 0)$

Require: Point $P = (X : Z)$ in Montgomery coordinates and theta null point $0 = (a : b)$
Ensure: Point $P = (\theta_0 : \theta_1)$ in theta coordinates

1. $\theta_0 \leftarrow X - Z$
2. $\theta_0 \leftarrow a \cdot \theta_0$
3. $\theta_1 \leftarrow X + Z$
4. $\theta_1 \leftarrow b \cdot \theta_1$
5. **return** $(\theta_0 : \theta_1)$
. ▷ Cost: 2M + 2a

D.1.2 On principally polarized abelian surfaces

Let A be a principally polarized abelian surface over $\mathbb{F}_{p^2}$. A level-2 theta structure on A is represented by projective coordinates $(x : y : z : w)$. When A is a Jacobian, these coordinates determine a point up to sign. When A is isomorphic to a product $E_1 \times E_2$, they encode a pair of elliptic points, again only up to the expected sign ambiguities. As in the elliptic-curve case, the whole coordinate system is determined by the theta null point $0_A = (a : b : c : d) := (x(0) : y(0) : z(0) : w(0))$. Different choices of theta structure on the same surface lead to different projective coordinate systems, related by explicit linear changes of variables. In the implementation, we fix one such theta structure once and for all on every intermediate codomain, so that all subsequent doubling and evaluation formulas are expressed in a uniform set of coordinates.

D.1.3 Product theta coordinates

Let $A = E_1 \times E_2$ be a product of Montgomery elliptic curves, and let

$$(\theta_0 : \theta_1) \in E_1, \qquad (\theta_0' : \theta_1') \in E_2$$

denote the level-2 theta coordinates of the two components. The associated product theta coordinates on A are $(x : y : z : w) = (\theta_0\theta_0' : \theta_1\theta_0' : \theta_0\theta_1' : \theta_1\theta_1')$. These are the coordinates naturally attached to the product theta structure. They are not, however, the coordinates in which the intermediate abelian surfaces of the chain are represented. The gluing step therefore requires an explicit change of basis from product theta coordinates to the fixed theta structure used on the codomain. Conversely, after the final splitting step, one returns from this fixed theta structure to a product theta structure in order to recover Montgomery models for the two elliptic factors.

D.2 Doubling formulas using theta coordinates

Let A be a principally polarized abelian surface with theta null point $0_A = (a : b : c : d)$. We write

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

for the Hadamard transform and componentwise squaring operator, respectively. The Hadamard transform costs 8a and $\mathcal{S}$ costs 4S.

From the theta null point one first derives the dual theta null point $(a' : b' : c' : d') := \mathcal{H}(a, b, c, d)$. For the canonical 2-isogeny attached to the theta structure, the dual-isogenous theta null point is $(\alpha^2 : \beta^2 : \gamma^2 : \delta^2) = \mathcal{H}(a^2 : b^2 : c^2 : d^2)$ These constants are exactly the quantities needed to express doubling in theta coordinates.

In the implementation, the cost of repeated doubling is reduced by precomputing the products of theta constants that occur systematically in the formulas. This is the purpose of ThetaPrecomp. Once these auxiliary values have been prepared, ThetaDBL applies a fixed sequence of squarings, Hadamard transforms, and rescalings to obtain the theta coordinates of $[2]P$. Since the same codomain theta null point is typically used for many successive doublings before the next isogeny step is taken, this precomputation is reused throughout each level of the chain.

Algorithm 38 ThetaPrecomp(0_A)

Require: Theta null point $0_A = (a : b : c : d)$
Ensure: Auxiliary constants consts used for arithmetic

1. $(A, B, C, D) \leftarrow \mathcal{H} \circ \mathcal{S}(a, b, c, d)$
2. $t_1 \leftarrow A \cdot B$
3. $t_2 \leftarrow C \cdot D$
4. $ABC \leftarrow t_1 \cdot C$
5. $ABD \leftarrow t_1 \cdot D$
6. $ACD \leftarrow t_2 \cdot A$
7. $BCD \leftarrow t_2 \cdot B$
8. $t_1 \leftarrow a \cdot b$
9. $t_2 \leftarrow c \cdot d$
10. $abc \leftarrow t_1 \cdot c$
11. $abd \leftarrow t_1 \cdot d$
12. $acd \leftarrow t_2 \cdot a$
13. $bcd \leftarrow t_2 \cdot b$
14. consts $\leftarrow \{abc, abd, acd, bcd, ABC, ABD, ACD, BCD\}$
15. **return** consts

▷ Cost: 4S + 12M (+8a)

Algorithm 39 ThetaDBL(P, consts)

Require: Theta coordinates P on A with theta null point $0_A = (a : b : c : d)$, and consts = ThetaPrecomp(0_A)
Ensure: Theta coordinates of $[2]P$

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

▷ Cost: 8S + 8M + 16a

D.3 Generic $(2, 2)$-isogeny computation

We now turn to the generic step of the chain. Let $\Phi : A \to B$ be a $(2, 2)$-isogeny between principally polarized abelian surfaces, with both domain and codomain represented in the fixed theta structure. The basic problem is to recover the theta null point of B and the inverse dual theta point and the associated quantities needed to evaluate Φ on arbitrary points, starting from torsion information above $\ker(\Phi)$. The implementation uses three codomain recovery routines, depending on how much torsion above the kernel is available.

- When compatible isotropic 8-torsion points are known, one uses the cheapest and most stable formulas, see Section D.3.1 and specifically GenericCodomainWith8Torsion.
- Near the end of a chain, only compatible 4-torsion may remain, in which case codomain recovery is still possible but requires square roots, see Section D.3.2 and specifically GenericCodomainWith4Torsion.
- At the last step, when one has only the kernel generators themselves, the codomain is recovered directly from the theta null point of the domain, see Section D.3.3 and specifically GenericCodomain.

- Regardless of which method is used for codomain computation, we may then use GenericEval to evaluate the isogeny, see Section D.3.4.

Throughout, we assume that the kernel is compatible with the theta structure on A. This compatibility is not an auxiliary cosmetic condition: it is the hypothesis under which the codomain theta structure is recovered by the formulas below. In a chain of $(2,2)$-isogenies, compatibility is enforced at the initial gluing step and then propagated from one level to the next by the shape checks on the transported torsion points.

D.3.1 With isotropic 8-torsion lying above the kernel

Assume that theta coordinates of two points $T_1'', T_2'' \in A[8]$ are known, with $\ker(\Phi) = \langle [4]T_1'' \rangle \oplus \langle [4]T_2'' \rangle$. This is the situation used throughout the main body of the chain whenever compatible 8-torsion above the next kernel is available.

From these two points, GenericCodomainWith8Torsion reconstructs three pieces of codomain quantities: $(\alpha : \beta : \gamma : \delta)$, $(\alpha^{-1} : \beta^{-1} : \gamma^{-1} : \delta^{-1})$, and $0_B = (a_2 : b_2 : c_2 : d_2)$. Here the first tuple is the dual isogenous theta null point, the second is its projective inverse, and the third is the theta null point of the codomain surface B itself.

The reason this case is the most efficient is that the images of T_1'' and T_2'' under $\mathcal{H} \circ \mathcal{S}$ already contain enough multiplicative information to determine the four dual theta coordinates up to the required projective scaling. No square root extraction is needed. In particular, once compatible 8-torsion is available, codomain recovery and subsequent evaluation both stay in a purely multiplicative regime.

Two issues must nevertheless be controlled when this routine is used repeatedly in a chain. First, the generic situation requires $\alpha\beta\gamma\delta \neq 0$. If one of the relevant projective factors vanishes unexpectedly, then the codomain has ceased to be generic; operationally, this means that the codomain has become non-generic. Second, even if the codomain is generic, the image of the transported torsion points must still define the correct compatible 4-torsion on the codomain. The purpose of the isotropy check is precisely to verify this second point.

Isotropy and compatibility checks. Let $P = \mathcal{H} \circ \mathcal{S}(T_1'')$ and $Q = \mathcal{H} \circ \mathcal{S}(T_2'')$, and let $I = (\alpha^{-1} : \beta^{-1} : \gamma^{-1} : \delta^{-1})$ be the inverse dual theta point on the codomain. The check performed in CheckIsotropic ensures that the images of the chosen 8-torsion points have exactly the form required to serve as compatible 4-torsion above the next kernel.

Concretely, the image of T_1'' must have the shape $(x : x : y : y)$, while the image of T_2'' must have the shape $(z : w : z : w)$. Equivalently, one checks the four relations

$$x_P \alpha^{-1} = y_P \beta^{-1}, \qquad z_P \gamma^{-1} = w_P \delta^{-1},$$

and

$$x_Q \alpha^{-1} = z_Q \gamma^{-1}, \qquad y_Q \beta^{-1} = w_Q \delta^{-1}.$$

When these relations hold, the transported points lie above the correct 2-torsion kernel on the codomain and remain compatible with the theta structure carried by B. This is exactly the information needed for the next level of the chain.

Algorithm 40 GenericCodomainWith8Torsion(T_1'', T_2'')

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

$x_{T_1''}, y_{T_1''}, z_{T_1''}, w_{T_1''} \leftarrow T_1''$
$x_{T_2''}, y_{T_2''}, z_{T_2''}, w_{T_2''} \leftarrow T_2''$
$(x\alpha, x\beta, y\gamma, y\delta) \leftarrow \mathcal{H} \circ \mathcal{S}(x_{T_1''}, y_{T_1''}, z_{T_1''}, w_{T_1''})$
$(z\alpha, w\beta, z\gamma, w\delta) \leftarrow \mathcal{H} \circ \mathcal{S}(x_{T_2''}, y_{T_2''}, z_{T_2''}, w_{T_2''})$
if $0 \in \{x\alpha, x\beta, z\alpha, w\beta, z\gamma, w\delta\}$ **then**
    raise Exception: ("`generic-codomain-8torsion` failed: unexpected splitting")
$x\alpha w\beta \leftarrow x\alpha \cdot w\beta$
$z\alpha x\beta \leftarrow z\alpha \cdot x\beta$
$\alpha \leftarrow z\alpha \cdot x\alpha w\beta$
$\beta \leftarrow w\beta \cdot z\alpha x\beta$
$\gamma \leftarrow z\gamma \cdot x\alpha w\beta$
$\delta \leftarrow w\delta \cdot z\alpha x\beta$
$z\gamma w\delta \leftarrow z\gamma \cdot w\delta$
$\alpha^{-1} \leftarrow x\beta \cdot z\gamma w\delta$
$\beta^{-1} \leftarrow x\alpha \cdot z\gamma w\delta$
$\gamma^{-1} \leftarrow \delta$
$\delta^{-1} \leftarrow \gamma$
$P \leftarrow (x\alpha : x\beta : y\gamma : y\delta)$
$Q \leftarrow (z\alpha : w\beta : z\gamma : w\delta)$
$I \leftarrow (\alpha^{-1} : \beta^{-1} : \gamma^{-1} : \delta^{-1})$
if not CheckIsotropic(P, Q, I) **then**
    raise Exception: ("`generic-codomain-8torsion` failed: P,Q not isotropic")
$(a_2, b_2, c_2, d_2) \leftarrow \mathcal{H}(\alpha, \beta, \gamma, \delta)$
$0_B \leftarrow (a_2 : b_2 : c_2 : d_2)$
return $(\alpha : \beta : \gamma : \delta),\ (\alpha^{-1} : \beta^{-1} : \gamma^{-1} : \delta^{-1}),\ 0_B$ ▷ Total cost (without checks): 8S + 9M + 24a

We emphasize that T_1'' and T_2'' are used only to recover the codomain. After that point, the chain proceeds with the codomain theta null point, its inverse dual theta point, and the transported points produced by the evaluation formulas.

Algorithm 41 CheckIsotropic(P, Q, I)

Require: Theta coordinates $P = (x_P, y_P, z_P, w_P)$ and $Q = (x_Q, y_Q, z_Q, w_Q)$, and inverse dual theta point $I = (\alpha^{-1}, \beta^{-1}, \gamma^{-1}, \delta^{-1})$
Ensure: Boolean indicating whether the corresponding 4-torsion points are isotropic

$t_1 \leftarrow x_P \cdot \alpha^{-1}$	$t_2 \leftarrow y_P \cdot \beta^{-1}$	**if** $t_1 \neq t_2$, **return false**
$t_1 \leftarrow z_P \cdot \gamma^{-1}$	$t_2 \leftarrow w_P \cdot \delta^{-1}$	**if** $t_1 \neq t_2$, **return false**
$t_1 \leftarrow x_Q \cdot \alpha^{-1}$	$t_2 \leftarrow z_Q \cdot \gamma^{-1}$	**if** $t_1 \neq t_2$, **return false**
$t_1 \leftarrow y_Q \cdot \beta^{-1}$	$t_2 \leftarrow w_Q \cdot \delta^{-1}$	**if** $t_1 \neq t_2$, **return false**

return true ▷ Cost: at most 8M

D.3.2 With isotropic 4-torsion lying above the kernel

Suppose now that compatible 8-torsion above the kernel is no longer available, but a point $T_1' \in A[4]$ is known such that $[2]T_1' \in \ker(\Phi)$. In that case, the codomain can still be recovered by GenericCodomainWith4Torsion.

The difference from the previous case is that the available torsion lifts no longer determines the dual theta coordinates multiplicatively without ambiguity. The algorithm therefore combines the quantities extracted from $\mathcal{H} \circ \mathcal{S}(T_1')$ with the domain invariants

$$(\alpha^2, \beta^2, \gamma^2, \delta^2) = \mathcal{H} \circ \mathcal{S}(a, b, c, d), \qquad 0_A = (a : b : c : d),$$

and recovers the codomain by extracting square roots at the appropriate stage. The output has the same form as before: $(\alpha : \beta : \gamma : \delta)$, $(\alpha^{-1} : \beta^{-1} : \gamma^{-1} : \delta^{-1})$, and 0_B.

This routine is used only when the available torsion has dropped from level 8 to level 4. It is therefore naturally confined to the last non-terminal part of the chain. Compared with the 8-torsion case, the formulas are more expensive and less uniform, but they still fit into the same downstream evaluation interface.

Algorithm 42 GenericCodomainWith4Torsion$(T_1', 0_A)$

Require: Theta coordinates of the order-4 point T_1' such that $[2]T_1' \in \ker(\Phi)$, and theta null point $0_A = (a : b : c : d)$
Ensure: Dual isogenous theta null point $(\alpha : \beta : \gamma : \delta)$, its inverse $(\alpha^{-1} : \beta^{-1} : \gamma^{-1} : \delta^{-1})$, and theta null point 0_B on B

$(x\alpha\beta, \_, x\gamma\delta, \_) \leftarrow \mathcal{H} \circ \mathcal{S}(x_{T_1'}, y_{T_1'}, z_{T_1'}, w_{T_1'})$
$(\alpha^2, \beta^2, \gamma^2, \delta^2) \leftarrow \mathcal{H} \circ \mathcal{S}(a, b, c, d)$
$\alpha\beta \leftarrow \sqrt{\alpha^2\beta^2}$ | $\alpha\gamma \leftarrow \sqrt{\alpha^2\gamma^2}$ | $\beta \leftarrow \alpha\beta \cdot \alpha\gamma$
$\delta^{-1} \leftarrow \beta \cdot x\gamma\delta$ | $\beta \leftarrow \beta \cdot x\alpha\beta$ | $\delta \leftarrow x\gamma\delta \cdot \alpha\beta \cdot \alpha^2$
$\alpha \leftarrow x\alpha\beta \cdot \alpha^2$ | $\gamma \leftarrow \alpha \cdot \gamma^2$ | $\alpha \leftarrow \alpha \cdot \alpha\gamma$
$\alpha^{-1} \leftarrow x\alpha\beta \cdot \delta^2$ | $\gamma^{-1} \leftarrow \alpha^{-1} \cdot \beta^2$ | $\alpha^{-1} \leftarrow \alpha^{-1} \cdot \gamma^2$
$\beta^{-1} \leftarrow \alpha^{-1} \cdot \alpha\beta$ | $\alpha^{-1} \leftarrow \alpha^{-1} \cdot \beta^2$ | $\gamma^{-1} \leftarrow \gamma^{-1} \cdot \alpha\gamma$
$\delta^{-1} \leftarrow \delta^{-1} \cdot \beta^2$
$(a_2, b_2, c_2, d_2) \leftarrow \mathcal{H}(\alpha, \beta, \gamma, \delta)$
$0_B \leftarrow (a_2 : b_2 : c_2 : d_2)$
return $(\alpha : \beta : \gamma : \delta),\ (\alpha^{-1} : \beta^{-1} : \gamma^{-1} : \delta^{-1}),\ 0_B$

D.3.3 With kernel generators only

At the final stage of the chain, it may happen that no higher torsion above the kernel remains available and one is left only with generators $T_1, T_2 \in A[2]$ of $\ker(\Phi)$. In that case, the codomain can still be reconstructed from the theta null point of the domain alone by GenericCodomain.

Operationally, this works because once the kernel is known to be compatible with the theta structure, the remaining codomain information is already encoded in

$$(\alpha^2, \beta^2, \gamma^2, \delta^2) = \mathcal{H} \circ \mathcal{S}(a, b, c, d), \qquad 0_A = (a : b : c : d).$$

Recovering the codomain from this information requires three square roots, so this is the most expensive of the three generic routines. For that reason, it is reserved for the terminal step of the chain.

Algorithm 43 GenericCodomain(0_A)

Require: Theta constants $0_A = (a : b : c : d)$
Ensure: Dual isogenous theta null point $(\alpha : \beta : \gamma : \delta)$, its inverse $(\alpha^{-1} : \beta^{-1} : \gamma^{-1} : \delta^{-1})$, and theta null point 0_B on B

$(\alpha^2, \beta^2, \gamma^2, \delta^2) \leftarrow \mathcal{H} \circ \mathcal{S}(a, b, c, d)$ | $\alpha \leftarrow \alpha^2$ | $\beta \leftarrow \alpha^2 \cdot \beta^2$
$\gamma \leftarrow \alpha^2 \cdot \gamma^2$ | $\delta \leftarrow \alpha^2 \cdot \delta^2$ | $\beta \leftarrow \sqrt{\beta}$
$\gamma \leftarrow \sqrt{\gamma}$ | $\delta \leftarrow \sqrt{\delta}$ | $\alpha^{-1} \leftarrow \gamma^2 \cdot \delta^2$
$\beta^{-1} \leftarrow \alpha^{-1} \cdot \beta$ | $\alpha^{-1} \leftarrow \alpha^{-1} \cdot \beta^2$ | $\gamma^{-1} \leftarrow \delta^2 \cdot \beta^2 \cdot \gamma$
$\delta^{-1} \leftarrow \gamma^2 \cdot \beta^2 \cdot \delta$ | $(a_2, b_2, c_2, d_2) \leftarrow \mathcal{H}(\alpha, \beta, \gamma, \delta)$ | $0_B \leftarrow (a_2 : b_2 : c_2 : d_2)$
return $(\alpha : \beta : \gamma : \delta),\ (\alpha^{-1} : \beta^{-1} : \gamma^{-1} : \delta^{-1}),\ 0_B$

D.3.4 Generic isogeny evaluation

Once the codomain of a generic $(2,2)$-isogeny has been recovered—whether by GenericCodomainWith8Torsion, GenericCodomainWith4Torsion, or GenericCodomain—the isogeny itself is eval-

uated in a uniform way. More precisely, suppose that the codomain theta null point 0_B and the inverse dual theta point

$$I = (\alpha^{-1} : \beta^{-1} : \gamma^{-1} : \delta^{-1})$$

are known. Then any point $P \in A$ can be evaluated under Φ using only its theta coordinates and I, as in GenericEval. The formula mirrors the doubling routine: one first applies $\mathcal{H} \circ \mathcal{S}$ to P, then rescales the four coordinates by the corresponding entries of I, and finally applies a Hadamard transform to obtain the theta coordinates of $\Phi(P)$. In particular, the evaluation step depends only on the inverse dual theta point, while the codomain theta null point is mainly needed to prepare the constants for future doublings on the new codomain.

Algorithm 44 GenericEval(P, I)

Input: Theta coordinates of P and inverse dual theta null point $I = (\alpha^{-1} : \beta^{-1} : \gamma^{-1} : \delta^{-1})$
Output: Theta coordinates of $\Phi(P)$

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

Translation action and change of basis. Before discussing the gluing step, one further ingredient is needed. On the product surface

$$E_1 \times E_2,$$

the natural product theta structure does not coincide with the fixed theta structure used on the glued codomain. We therefore need an explicit linear change of coordinates from product theta coordinates to the chosen theta structure.

This change of basis is built from the action of translation by suitable 4-torsion points on the elliptic factors. ActionByTranslation computes the corresponding 2×2 translation matrices on each factor, and ThetaChangeOfBasis assembles them into a 4×4 matrix N, which is then reused throughout the gluing step. The purpose of this matrix is purely geometric: it transports points from product theta coordinates into the fixed theta structure in which the codomain and all later intermediate surfaces are represented.

Algorithm 45 ActionByTranslation(P, Q)

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

$P' = (P'_1, P'_2) \leftarrow [2]P$
$Q' = (Q'_1, Q'_2) \leftarrow [2]Q$
for i from 1 up to 2 **do**
    Write $P_i = (X_i^{(P)} : Z_i^{(P)})$ and $Q_i = (X_i^{(Q)} : Z_i^{(Q)})$
    Write $P'_i = (U_i^{(P)} : W_i^{(P)})$ and $Q'_i = (U_i^{(Q)} : W_i^{(Q)})$
    $\delta_i^{(P)} \leftarrow W_i^{(P)} X_i^{(P)} - U_i^{(P)} Z_i^{(P)}$
    $\delta_i^{(Q)} \leftarrow W_i^{(Q)} X_i^{(Q)} - U_i^{(Q)} Z_i^{(Q)}$
Compute the inverses of

$$\delta_1^{(P)},\ \delta_2^{(P)},\ \delta_1^{(Q)},\ \delta_2^{(Q)},\ Z_1^{(P)},\ Z_2^{(P)},\ Z_1^{(Q)},\ Z_2^{(Q)}$$

by batched inversion
Initialize mats $\leftarrow [\,]$
pts $\leftarrow [P, Q]$
for i from 1 up to 2 **do**
    for j from 1 up to 2 **do**
        $R \leftarrow \text{pts}[j]$
        $M_{0,0} \leftarrow -U_i^{(R)} Z_i^{(R)} \cdot (\delta_i^{(R)})^{-1}$
        $M_{0,1} \leftarrow -W_i^{(R)} Z_i^{(R)} \cdot (\delta_i^{(R)})^{-1}$
        $M_{1,0} \leftarrow U_i^{(R)} X_i^{(R)} \cdot (\delta_i^{(R)})^{-1} - X_i^{(R)} \cdot (Z_i^{(R)})^{-1}$
        $M_{1,1} \leftarrow -M_{0,0}$
        $M \leftarrow (M_{u,v})_{0 \le u,v \le 1}$
        Append M to mats
return mats

D.4 Gluing $(2, 2)$-isogeny

We now describe the first step of the chain, namely a gluing isogeny $\Phi : E_1 \times E_2 \to A$, The input contains 8-torsion points $T''_1, T''_2 \in E_1 \times E_2$ such that $\ker(\Phi) = \langle [4]T''_1 \rangle \oplus \langle [4]T''_2 \rangle$.

The computation first changes from product theta coordinates to the chosen theta structure used in the gluing step, then computes the codomain theta null point, and finally evaluates points through Φ.

D.4.1 Gluing $(2, 2)$-isogeny codomain computation

The first task is to move points from the product theta structure of $E_1 \times E_2$ to the theta structure used on A. Starting from doubling the input $T'_1 = [2]T''_1, T'_2 = [2]T''_2$ to obtain compatible 4-torsion points and then calls ThetaChangeOfBasis on T'_1, T'_2. This routine uses the translation matrices computed by ActionByTranslation and returns a 4×4 change-of-basis matrix N.

The matrix N changes product theta coordinates on $E_1 \times E_2$ to the chosen theta structure. Thus, for a point $P = (P_1, P_2)$, the routine ProductToTheta first forms product theta coordinates from the dimension one theta coordinates of P_1 and P_2, and then applies N. When the input points are given in Montgomery coordinates, the component coordinates are first obtained using MontgomeryToTheta.

Then algorithm applies N to the two 8-torsion points, thereby changing from the product theta structure to the chosen theta structure. Applying $\mathcal{H} \circ \mathcal{S}$ to them gives the quantities used to recover the dual theta

null point $(\alpha : \beta : \gamma : 0)$, its projective inverse $(\alpha^{-1} : \beta^{-1} : \gamma^{-1} : 0)$, and the codomain theta null point $0_A = \mathcal{H}(\alpha, \beta, \gamma, 0)$.

The gluing codomain is distinguished from the generic case by the fact that, with our choice of theta structure, its dual theta null point of codomain has the special shape $(\alpha : \beta : \gamma : 0)$. Thus gluing is not handled by a completely separate formalism; rather, it appears as a structured degenerate case of codomain recovery, in which the last dual coordinate vanishes by design.

Besides these quantities, the gluing step also returns an auxiliary point $J = (x : x : y : y) = \widehat{\Phi(T_1'')}$. Both J and N are reused in GluingEval: N is used to map input points in the chosen theta structure.

Three checks are essential here. First, after applying $\mathcal{H} \circ \mathcal{S}$, the last coordinate must vanish; otherwise the computation is not in the gluing configuration dictated by the chosen theta structure. Second, the remaining projective factors used to form α, β, γ and their inverses must be non-zero. Third, the doubled points $[2]T_1''$ and $[2]T_2''$ must be isotropic and compatible with the chosen theta structure. These conditions ensure that the chain is started in the correct coordinate system and that the generic routines used afterwards will apply without further change of model.

Algorithm 46 ThetaChangeOfBasis(P, Q)

Input: Points $P = (P_1, P_2)$ and $Q = (Q_1, Q_2)$ of order 4 in $E_1 \times E_2$ such that the gluing kernel is $\langle [2]P, [2]Q \rangle$
Output: A 4×4 change-of-basis matrix N

if P_1, P_2, Q_1, Q_2 do not have order 4 or $[2]P_1 = [2]P_2$ or $[2]Q_1 = [2]Q_2$ **then**
  raise Exception: ("`theta-change-of-basis` failed: gluing kernel is diagonal or not isotropic")
$(G, G', H, H') \leftarrow$ ActionByTranslation(P, Q)
$t_1 \leftarrow G_{0,0} \cdot H_{0,0} + G_{0,1} \cdot H_{1,0}$
$t_2 \leftarrow G_{1,0} \cdot H_{0,0} + G_{1,1} \cdot H_{1,0}$
$t_3 \leftarrow G'_{0,0} \cdot H'_{0,0} + G'_{0,1} \cdot H'_{1,0}$
$t_4 \leftarrow G'_{1,0} \cdot H'_{0,0} + G'_{1,1} \cdot H'_{1,0}$
$N_{0,0} \leftarrow G_{0,0} \cdot G'_{0,0} + H_{0,0} \cdot H'_{0,0} + t_1 \cdot t_3 + 1$
$N_{0,1} \leftarrow G_{0,0} \cdot G'_{1,0} + H_{0,0} \cdot H'_{1,0} + t_1 \cdot t_4$
$N_{0,2} \leftarrow G_{1,0} \cdot G'_{0,0} + H_{1,0} \cdot H'_{0,0} + t_2 \cdot t_3$
$N_{0,3} \leftarrow G_{1,0} \cdot G'_{1,0} + H_{1,0} \cdot H'_{1,0} + t_2 \cdot t_4$
$N_{1,0} \leftarrow H'_{0,0} \cdot N_{0,0} + H'_{0,1} \cdot N_{0,1}$
$N_{1,1} \leftarrow H'_{1,0} \cdot N_{0,0} + H'_{1,1} \cdot N_{0,1}$
$N_{1,2} \leftarrow H'_{0,0} \cdot N_{0,2} + H'_{0,1} \cdot N_{0,3}$
$N_{1,3} \leftarrow H'_{1,0} \cdot N_{0,2} + H'_{1,1} \cdot N_{0,3}$
$N_{2,0} \leftarrow G_{0,0} \cdot N_{0,0} + G_{0,1} \cdot N_{0,2}$
$N_{2,1} \leftarrow G_{0,0} \cdot N_{0,1} + G_{0,1} \cdot N_{0,3}$
$N_{2,2} \leftarrow G_{1,0} \cdot N_{0,0} + G_{1,1} \cdot N_{0,2}$
$N_{2,3} \leftarrow G_{1,0} \cdot N_{0,1} + G_{1,1} \cdot N_{0,3}$
$N_{3,0} \leftarrow G_{0,0} \cdot N_{1,0} + G_{0,1} \cdot N_{1,2}$
$N_{3,1} \leftarrow G_{0,0} \cdot N_{1,1} + G_{0,1} \cdot N_{1,3}$
$N_{3,2} \leftarrow G_{1,0} \cdot N_{1,0} + G_{1,1} \cdot N_{1,2}$
$N_{3,3} \leftarrow G_{1,0} \cdot N_{1,1} + G_{1,1} \cdot N_{1,3}$
$N \leftarrow (N_{i,j})_{0 \le i,j \le 3}$
return N

Algorithm 47 ProductToTheta(pts, N)

Input: List of points pts, where for $P \in$ pts we have $P = (P_1, P_2) \in E_1 \times E_2$, and change-of-basis matrix N

Output: The corresponding points in theta coordinates on $A \cong E_1 \times E_2$

eval_pts $\leftarrow [\,]$
$L \leftarrow \#$pts
for j from 1 up to L **do**
    $P = (P_1, P_2) \leftarrow$ pts$[j]$
    Write $P_1 = (\theta_0 : \theta_1)$ and $P_2 = (\theta'_0 : \theta'_1)$
    $x \leftarrow \theta_0 \cdot \theta'_0$
    $y \leftarrow \theta_0 \cdot \theta'_1$
    $z \leftarrow \theta_1 \cdot \theta'_0$
    $w \leftarrow \theta_1 \cdot \theta'_1$
    $P' \leftarrow N \cdot (x : y : z : w)$
    Append P' to eval_pts
return eval_pts

Algorithm 48 GluingCodomain(T''_1, T''_2)

Input: 8-torsion points $T''_1, T''_2 \in E_1 \times E_2$ with $\ker(\Phi) = \langle [4]T''_1 \rangle \oplus \langle [4]T''_2 \rangle$

Output: Dual isogenous theta null point $(\alpha : \beta : \gamma : 0)$, its inverse $(\alpha^{-1} : \beta^{-1} : \gamma^{-1} : 0)$, theta null point 0_A on A, the dual theta point $J = \widehat{\Phi(T''_1)}$, and the basis-change matrix N

$T'_1 \leftarrow [2](T''_1)$
$T'_2 \leftarrow [2](T''_2)$
$N \leftarrow$ ThetaChangeOfBasis(T'_1, T'_2)
$[P_1, P_2] \leftarrow$ ProductToTheta$([T''_1, T''_2], N)$
$x_1, y_1, z_1, w_1 \leftarrow P_1$
$x_2, y_2, z_2, w_2 \leftarrow P_2$
$(X_1, Y_1, Z_1, W_1) \leftarrow \mathcal{H} \circ \mathcal{S}(x_1, y_1, z_1, w_1)$
$(X_2, Y_2, Z_2, W_2) \leftarrow \mathcal{H} \circ \mathcal{S}(x_2, y_2, z_2, w_2)$
if $W_1 \neq 0$ or $W_2 \neq 0$ **then**
    raise Exception: ("`gluing-codomain` failed: last coordinate is non-zero")
if $X_1 = 0$ or $X_2 = 0$ or $Y_1 = 0$ or $Z_2 = 0$ **then**
    raise Exception: ("`gluing-codomain` failed: projective factors are zero")
$\alpha \leftarrow X_1 \cdot X_2$
$\beta \leftarrow Y_1 \cdot X_2$
$\gamma \leftarrow X_1 \cdot Z_2$
$\alpha^{-1} \leftarrow Y_1 \cdot Z_2$
$\beta^{-1} \leftarrow \gamma$
$\gamma^{-1} \leftarrow \beta$
$x \leftarrow X_1 \cdot \alpha^{-1}$
$y \leftarrow Z_1 \cdot \gamma^{-1}$
$J \leftarrow (x : x : y : y)$
if $(Y_1 \cdot \beta^{-1} \neq x)$ **or** $(X_2 \cdot \alpha^{-1} \neq Y_2 \cdot \beta^{-1})$ **then**
    raise Exception: ("`gluing-codomain` failed: $[2]T''_1$ and $[2]T''_2$ are not isotropic")
$(a_2, b_2, c_2, d_2) \leftarrow \mathcal{H}(\alpha, \beta, \gamma, 0)$
$0_A \leftarrow (a_2 : b_2 : c_2 : d_2)$
return $(\alpha : \beta : \gamma : 0),\ (\alpha^{-1} : \beta^{-1} : \gamma^{-1} : 0),\ 0_A,\ J,\ N$

D.4.2 Gluing $(2,2)$-isogeny evaluation

A general point $P = (P_1, P_2) \in E_1 \times E_2$ is then evaluated by GluingEval. It uses the same change of coordinates as in codomain computation, but it cannot be reduced to GenericEval. Indeed, the gluing codomain has a zero dual theta-null coordinate, so the generic evaluation formula would leave one dual coordinate of $\Phi(P)$ undetermined. The role of GluingEval is therefore to recover the missing information needed to compute the theta coordinates of $\Phi(P)$.

To do this, the algorithm uses the 8-torsion T_1'' already used in GluingCodomain. On each elliptic factor, ADDComponents computes the local data associated with P_i and T_i. These local data encode the contribution of both P and the translated point $P + T_1''$. The algorithm then combines them into two four-tuples, applies the change-of-basis matrix N, and uses the auxiliary point J to fix the remaining projective scaling.

There is also a special case in which the evaluation is simpler, namely when the input point has the form $(P_1, 0)$ or $(0, P_2)$. In that situation, GluingEvalSpecial evaluates the point using only the inverse dual theta point and the basis-change matrix. This shortcut is useful for QIMEN-PRISM.

Algorithm 49 GluingEval(P, T_1'', J, N)

Input: Point $P \in E_1 \times E_2$, an 8-torsion point T_1'' such that $[4]T_1'' \in \ker(\Phi)$, the point $J = (x : x : y : y)$, and the change-of-basis matrix N returned by GluingCodomain

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

$P_1, P_2 \leftarrow P$
$T_1, T_2 \leftarrow T_1''$
$x, y \leftarrow J$
$u_1, v_1, z_1 \leftarrow$ ADDComponents(P_1, T_1, E_1)
$u_2, v_2, z_2 \leftarrow$ ADDComponents(P_2, T_2, E_2)
$U \leftarrow (u_1u_2 + v_1v_2,\ u_1z_2,\ z_1u_2,\ z_1z_2)$
$V \leftarrow (v_1u_2 + u_1v_2,\ v_1z_2,\ z_1v_2,\ 0)$
$U \leftarrow N \cdot U$
$V \leftarrow N \cdot V$
$U \leftarrow \mathcal{S}(U)$
$V \leftarrow \mathcal{S}(V)$
$X_\pm, Y_\pm, Z_\pm, W_\pm \leftarrow \mathcal{H}(U - V)$
$X_{\Phi(P)}, Y_{\Phi(P)}, Z_{\Phi(P)}, W_{\Phi(P)} \leftarrow X_\pm \cdot y,\ Y_\pm \cdot y,\ Z_\pm \cdot x,\ W_\pm \cdot x$
$(x_{\Phi(P)}, y_{\Phi(P)}, z_{\Phi(P)}, w_{\Phi(P)}) \leftarrow \mathcal{H}(X_{\Phi(P)}, Y_{\Phi(P)}, Z_{\Phi(P)}, W_{\Phi(P)})$
return $(x_{\Phi(P)} : y_{\Phi(P)} : z_{\Phi(P)} : w_{\Phi(P)})$

Algorithm 50 GluingEvalSpecial(P, I, N)

Input: Point $P \in E_1 \times E_2$ of the form $(P_1, 0)$ or $(0, P_2)$, the inverse dual theta point $I = (\alpha^{-1} : \beta^{-1} : \gamma^{-1} : 0)$, and the change-of-basis matrix N

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

$[\widetilde{P}] \leftarrow$ ProductToTheta$([P], N)$
$x_P, y_P, z_P, w_P \leftarrow \widetilde{P}$
$\alpha^{-1}, \beta^{-1}, \gamma^{-1}, \_ \leftarrow I$
$(X_P, Y_P, Z_P, 0) \leftarrow \mathcal{H} \circ \mathcal{S}(x_P, y_P, z_P, w_P)$
$(X_{\Phi(P)}, Y_{\Phi(P)}, Z_{\Phi(P)}) \leftarrow X_P \cdot \alpha^{-1},\ Y_P \cdot \beta^{-1},\ Z_P \cdot \gamma^{-1}$
$(x_{\Phi(P)}, y_{\Phi(P)}, z_{\Phi(P)}, w_{\Phi(P)}) \leftarrow \mathcal{H}(X_{\Phi(P)}, Y_{\Phi(P)}, Z_{\Phi(P)}, 0)$
return $(x_{\Phi(P)} : y_{\Phi(P)} : z_{\Phi(P)} : w_{\Phi(P)})$

D.5 Splitting change of coordinates

We now turn to the final transition of the chain. At this stage one has computed a last generic $(2,2)$-isogeny $\Phi : A \to B$, where the codomain B is known to be isomorphic to a product of elliptic curves, but is still expressed in the fixed theta structure rather than in product theta coordinates. This final stage should therefore be viewed in two parts.

1. First, one computes the last generic codomain exactly as in the preceding subsection, using one of the three generic codomain routines depending on the torsion information still available.

2. Second, one computes an explicit isomorphism that sends the theta null point of this codomain to the product theta structure of $E_1 \times E_2$. This is handled by SplittingIsomorphism, again in two parts.

   - SplittingIsomorphism first determines the unique index pair (i,j) for which $U_{i,j}(0_A) = 0$, where the quantities $U_{i,j}$ are level-$(2,2)$ theta expressions built from the theta null point, using the subalgorithm GetIndexSplitting. Explicitly,

   $$U_{i,j}(0_A) = \sum_{t=0}^{3} \chi(i,t)\, 0_A[t]\, 0_A[j \oplus t],$$

   where $\oplus$ denotes bitwise XOR and χ is the corresponding sign character.
   - Once the correct pair (i,j) has been found, SplittingIsomorphism selects the corresponding precomputed matrix $M \in \mathrm{Mat}_{4\times 4}$, whose action transports the current theta structure back to the product theta structure.

After this change of coordinates, the codomain is genuinely represented as a product. One may then recover the Montgomery coefficients of the two elliptic factors from the transformed theta null point using ThetaToProduct, and convert theta product coordinates of evaluated points back to Montgomery coordinates on each factor via ThetaProductPointToMontgomery.

Algorithm 51 GetIndexSplitting(0_A)

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

inds $\leftarrow \{(0,0),(0,1),(0,2),(0,3),(1,0),(1,2),(2,0),(2,1),(3,0),(3,3)\}$
count $\leftarrow 0$
got_index $\leftarrow$ **false**
for k from 1 up to #inds **do**
    $(i,j) \leftarrow$ inds$[k]$
    $U \leftarrow \sum_{t=0}^{3} \chi(i,t) \cdot 0_A[t] \cdot 0_A[j \oplus t]$
    if $U = 0$ **then**
        count $\leftarrow$ count $+ 1$
        ind $\leftarrow (i,j)$
        got_index $\leftarrow$ **true**
if count $\neq 1$ or **not** got_index **then**
    raise Exception: ("`get-index-splitting` failed: zero or multiple zero indices found")
return ind

Algorithm 52 SplittingIsomorphism(0_A)

Require: Theta null point 0_A of $A \cong E_1 \times E_2$
Ensure: Isomorphism matrix M whose action on 0_A gives back the theta null point associated with the product theta structure

1. $(i,j) \leftarrow$ GetIndexSplitting(0_A)

$(i,j) = (0,0)$:

2. $M \leftarrow \begin{pmatrix} 1 & \sqrt{-1} & 1 & \sqrt{-1} \\ 1 & -\sqrt{-1} & -1 & \sqrt{-1} \\ 1 & -\sqrt{-1} & -1 & -\sqrt{-1} \\ -1 & \sqrt{-1} & -1 & \sqrt{-1} \end{pmatrix}$

$(i,j) = (1,0)$:

3. $M \leftarrow \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & -1 & 1 \\ 1 & 1 & -1 & -1 \\ -1 & 1 & -1 & 1 \end{pmatrix}$

$(i,j) = (2,0)$:

4. $M \leftarrow \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & -1 & -1 & 1 \\ -1 & -1 & 1 & 1 \end{pmatrix}$

$(i,j) = (3,0)$:

5. $M \leftarrow \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ -1 & 1 & 1 & -1 \end{pmatrix}$

$(i,j) = (0,1)$:

6. $M \leftarrow \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & -1 & 0 & 0 \end{pmatrix}$

$(i,j) = (2,1)$:

7. $M \leftarrow \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & 1 & -1 & -1 \end{pmatrix}$

$(i,j) = (0,2)$:

8. $M \leftarrow \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & -1 & 0 \end{pmatrix}$

$(i,j) = (1,2)$:

9. $M \leftarrow \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$

$(i,j) = (0,3)$:

10. $M \leftarrow \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix}$

$(i,j) = (3,3)$:

11. $M \leftarrow \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$

12. **return** M

Algorithm 53 ThetaToProduct(0_A)

Require: Theta null point $0_A = (a : b : c : d)$ of PPAS A with theta product structure
Ensure: Montgomery coefficients $(A_1 : C_1)$ and $(A_2 : C_2)$ of E_1 and E_2, respectively, such that $A \cong E_1 \times E_2$

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

Algorithm 54 ThetaProductPointToMontgomery$(P, 0_A)$

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

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

D.6 Computing a $(2,2)$-isogeny chain between products of elliptic curves

We are now ready to describe the full computation of a $(2,2)$-isogeny chain

$$\Phi = \Phi_e \circ \cdots \circ \Phi_1 : E_1 \times E_2 \longrightarrow E_3 \times E_4.$$

The computation Isogeny22Chain has a clear three-phase structure:

1. a gluing step from a product surface to a theta-coordinate model of a principally polarized abelian surface,
2. a sequence of generic $(2,2)$-isogenies entirely inside that model, and
3. a final splitting step returning to a product of elliptic curves.

We organize the algorithmic presentation according to this structure. Besides the main routine Isogeny22Chain, we first give one auxiliary routine, ChainStrategyComputation, and then three phase subroutines: GluingStep, GenericStep, and SplittingStep.

Throughout this subsection, we use the torsion-rich input convention used by the implementation. Namely, the routine takes as input two points $P, Q \in E_1 \times E_2$ of order 2^{e+2}. Thus the input provides two extra layers of 2-power torsion above the actual kernel. This makes compatible 8-torsion available for the chain computation.

The subroutines below are used as follows.

1. The auxiliary routine ChainStrategyComputation manages the strategy stack. It repeatedly applies the current doubling routine until the stack top gives the 8-torsion input required for the next codomain computation.
2. GluingStep performs the initial gluing step. It uses ChainStrategyComputation to first descend to compatible 8-torsion above the initial kernel, computes the gluing codomain by GluingCodomain, and transports all pending points to theta coordinates by GluingEval or GluingEvalSpecial.
3. Once the first codomain has been obtained, the remaining chain is computed entirely in theta coordinates. GenericStep computes all intermediate generic $(2,2)$-isogenies in theta coordinates. At each intermediate level, it uses ThetaDBL inside ChainStrategyComputation, recovers the next codomain by GenericCodomainWith8Torsion, and evaluates all pending points by GenericEval.
4. Finally, SplittingStep applies SplittingIsomorphism and transforms the codomain back to product theta coordinates. The two Montgomery factors are recovered, and all evaluated points are converted back to Montgomery coordinates.

Algorithm 55 ChainStrategyComputation(`strat_pts`, `orders`)

Input: Lists `strat_pts` and `orders` of the same length, whose last entries represent the current stack top
Output: Updated lists `strat_pts` and `orders` whose last point pair is an 8-torsion point

while `orders`$[k] \neq 1$ **do**
  $k \leftarrow k + 1$
  if `orders`$[k-1] \geq 16$ **then**
    $n \leftarrow \lfloor$`orders`$[k-1]/2 \rfloor$
  else
    $n \leftarrow$ `orders`$[k-1] - 1$
  $(R, S) \leftarrow$ `strat_pts`$[k-1]$
  for j from 1 up to n **do**
    $(R, S) \leftarrow \mathrm{DBL}(R, S)$
  `strat_pts`$[k] \leftarrow (R, S)$
  `orders`$[k] \leftarrow$ `orders`$[k-1] - n$
return `strat_pts`, `orders`

Algorithm 56 GluingStep(P, Q, e, pts)

Input: Points $P, Q \in E_1 \times E_2$ of order 2^{e+2}, and an array pts of points of the form $(R_1, 0)$ or $(0, R_2)$
Output: The first codomain theta null point 0_A, doubling constants, transported evaluation points, and the remaining strategy state

strat_pts $\leftarrow [(P, Q)]$
orders $\leftarrow [e]$
$k \leftarrow 0$
(strat_pts, orders) $\leftarrow$ ChainStrategyComputation(strat_pts, orders, k)
$(T_1'', T_2'') \leftarrow$ strat_pts$[k]$
$(\_, I, 0_A, J, N) \leftarrow$ GluingCodomain(T_1'', T_2'')
pts $\leftarrow [\,$GluingEvalSpecial$(R, I, N) \mid R \in$ pts$\,]$
for i from 0 up to $k-1$ **do**
  strat_pts$[i] \leftarrow$ GluingEval(strat_pts$[i], T_1'', J, N$)
  orders$[i] \leftarrow$ orders$[i] - 1$
$k \leftarrow k - 1$
consts $\leftarrow$ ThetaPrecomp(0_A)
return 0_A, consts, pts, strat_pts, orders

Algorithm 57 GenericStep$(0_A, \mathrm{consts}, \mathrm{pts}, \mathrm{strat_pts}, \mathrm{orders})$

Input: Current theta null point 0_A, doubling constants consts, points need to be evaluated pts, and kernel stack strat_pts, orders
Output: Final theta null point 0_A and evaluated points pts

while $k \geq 0$ and orders$[k] \neq 0$ **do**
  (strat_pts, orders) $\leftarrow$ ChainStrategyComputation(strat_pts, orders) ▷ using DBL = (ThetaDBL(_, consts)
  $(T_1'', T_2'') \leftarrow$ strat_pts$[k]$
  $(0_B', I, 0_B) \leftarrow$ GenericCodomainWith8Torsion(T_1'', T_2'')
  pts $\leftarrow [\,$GenericEval$(R, I) \mid R \in$ pts$\,]$
  consts $\leftarrow$ ThetaPrecomp(0_B)
  for i from 0 up to $k-1$ **do**
    strat_pts$[i] \leftarrow$ GenericEval(strat_pts$[i], I$)
    orders$[i] \leftarrow$ orders$[i] - 1$
  $k \leftarrow k - 1$
return 0_B, pts

Algorithm 58 SplittingStep$(0_A, \text{pts})$

Input: Theta null point 0_A of a surface isomorphic to a product, and evaluated points pts
Output: Product codomain $E_3 \times E_4$ and evaluated points in Montgomery coordinates on the two factors

$M \leftarrow$ SplittingIsomorphism(0_A)
$0_A \leftarrow M \cdot 0_A$
$\text{pts} \leftarrow [\, M \cdot R \mid R \in \text{pts}\,]$
$(A_3 : C_3), (A_4 : C_4) \leftarrow$ ThetaToProduct(0_A)
$\text{pts} \leftarrow [\,$ ThetaProductPointToMontgomery$(R, 0_A) \mid R \in \text{pts}\,]$
Let E_3, E_4 be the elliptic curves defined by Montgomery coefficients $(A_3 : C_3)$ and $(A_4 : C_4)$
return $E_3 \times E_4, \text{pts}$

Having described the auxiliary routine and the three phase subroutines, we now assemble them into the main routine Isogeny22Chain. This wrapper follows the three-phase structure described above: it first calls GluingStep, then GenericStep, and finally SplittingStep.

Algorithm 59 Isogeny22Chain(P, Q, pts)

Input: Points $P, Q \in E_1 \times E_2$ of order 2^{e+2}, and an array pts of points of the form $(R_1, 0)$ or $(0, R_2)$
Output: The codomain $E_3 \times E_4$ of the chain with $\ker(\Phi) = \langle [4]P, [4]Q \rangle$, together with the evaluated points $[\Phi(R) \mid R \in \text{pts}]$

$(0_A, \text{consts}, \text{pts}, \text{strat\_pts}, \text{orders}) \leftarrow$ GluingStep(P, Q, e, pts)
$(0_A, \text{pts}) \leftarrow$ GenericStep$(0_A, \text{consts}, \text{pts}, \text{strat_pts}, \text{orders})$
$(E_3 \times E_4, \text{pts}) \leftarrow$ SplittingStep$(0_A, \text{pts})$
return $E_3 \times E_4, \text{pts}$

CHAPTER E

Pairing computation (implementation details)*

In this section, we describe our use of cubical arithmetic in the implementation to compute pairings. Our presentation follows the cubical arithmetic of Pope, Reijnders, Robert, Sferlazza, and Smith [PRR+25]. Cubical arithmetic is slightly different from differential arithmetic, and we therefore denote this representation of $P \in E$ as a *cubical point* with a tilde, $\widetilde{P}$. Starting from cubical points

$$\widetilde{P} = (x(P) : z(P)), \quad \widetilde{Q} = (x(Q) : z(Q)), \quad \widetilde{P+Q} = (x(P+Q) : z(P+Q)), \quad \widetilde{0} = (1:0)$$

a cubical ladder produces cubical points for $[n]P$ and $[n]P + Q$, which we call *affine lifts* of such points. When $[n]P$ is the point at infinity or a 2-torsion point, these affine lifts $\widetilde{[n]P}$ and $\widetilde{[n]P+Q}$ differ from the starting points $\widetilde{0}$ and $\widetilde{Q}$ by scalar factors λ, λ'. The crux of cubical pairings is that such ratios λ/λ' contain enough information to compute Tate or Weil pairings. Thus, we can compute these pairings from the projective scaling information carried by the ladder output,[1] rather than from a Miller function evaluation.

E.1 Cubical arithmetic

Cubical pairing computations use the same x-only primitives as the Montgomery ladder, although cubical addition is slightly different from differential addition to keep track of the correct affine scaling. We use four basic routines.

- CubicalDbl to compute the cubical double $\widetilde{2P}$ of a cubical point $\widetilde{P}$,
- CubicalDiffAdd to compute $\widetilde{P+Q}$ given $\widetilde{P}$, $\widetilde{Q}$, and $\widetilde{P-Q}$,
- CubicalTranslate for the final doubling in the cubical ladder, which exists for technical reasons,
- CubicalRatio to recover the ratio λ between two affine lifts of the same point $P \in E$.

Algorithm 60 CubicalDbl$(E, \widetilde{P})$

Input: Montgomery curve $E : y^2 = x^3 + Ax^2 + x$, cubical point $\widetilde{P} = (X(P) : Z(P))$
Output: Cubical double $2\widetilde{P} = (X_2 : Z_2)$

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

[1] We omit the explicit division by 4 in the output of cubical differential addition. For reduced Tate pairings over $\mathbb{F}_{p^2}$, these factors are removed by the final exponentiation. For Weil pairings, they can be computed as the ratio of two squared Tate pairings.

Algorithm 61 CubicalDiffAdd($E, \widetilde{P}, \widetilde{Q}, x(P-Q)$)

Input: Montgomery curve $E : y^2 = x^3 + Ax^2 + x$; cubical points $\widetilde{P} = (X(P) : Z(P))$, $\widetilde{Q} = (X(Q) : Z(Q))$ and the x-coordinate of their difference $x(P-Q)$

Output: Cubical differential addition $\widetilde{P+Q} = (X_2, Z_2)$

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

Algorithm 62 CubicalTranslate($\widetilde{P}, \widetilde{T}$)

Input: Cubical Montgomery coordinates $\widetilde{P} = (X(P) : Z(P))$ and $\widetilde{T} = (X(T) : Z(T))$ where $T = (X(T) : Z(T))$ is a 2-torsion point

Output: Cubical translation $\widetilde{P+T} = (X(P+T), Z(P+T))$

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

Algorithm 63 CubicalRatio($\widetilde{P_1}, \widetilde{P_2}$)

Input: Cubical Montgomery coordinates $\widetilde{P_1} = (X(P_1) : Z(P_1))$, $\widetilde{P_2} = (X(P_2) : Z(P_2))$ such that $P_1 = (X(P_1) : Z(P_1)) = P_2 = (X(P_2) : Z(P_2))$

Output: Ratio λ such that $X(P_2) = \lambda X(P_1)$ and $Z(P_2) = \lambda Z(P_1)$

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

E.2 Even-degree pairings

When the pairing order $N = 2n$ is even, the square of the pairing loses one bit of information. In this case, one replaces the degree-N ladder by a degree-n ladder and then applies an affine translation by the 2-torsion point $[n]P$. In this specification we only consider the case that $N = 2^e$.

Algorithm 64 CubicalLadderPowertwo$(E, e, \widetilde{P+Q}, \widetilde{P}, x(Q))$

Input: Montgomery curve $E : y^2 = x^3 + Ax^2 + x$; integer e; cubical points $\widetilde{P+Q} = (X(P+Q) : Z(P+Q))$, $\widetilde{P} = (X(P) : Z(P))$; the x-coordinate of Q
Output: Cubical points $[2^e]\widetilde{P}$ and $[2^e]\widetilde{P} + \widetilde{Q}$

$n_{PQ} \leftarrow \widehat{P+Q}$
$n_P \leftarrow \widetilde{P}$
for $k \leftarrow 1$ **to** e **do**
    $n_{PQ} \leftarrow$ CubicalDiffAdd$(E, n_{PQ}, n_P, x(Q))$
    $n_P \leftarrow$ CubicalDbl(E, n_P)
return (n_P, n_{PQ})

Algorithm 65 Tate$(E, e, x(P), x(Q), x(P+Q))$

Input: Supersingular Montgomery curve $E : y^2 = x^3 + Ax^2 + x$; integer e; x-coordinates of points $P, Q, P+Q \in E(\mathbb{F}_{p^2})$ with $2^e P = 0_E$
Output: Reduced Tate pairing $t_{2^e}(P, Q) \in \mu_{2^e}$

$(n_P, n_{PQ}) \leftarrow$ CubicalLadderPowertwo$(E, e-1, (x(P+Q), 1), (x(P), 1), x(Q))$
$\widetilde{O} \leftarrow$ CubicalTranslate(n_P, n_P)
$\widetilde{Q'} \leftarrow$ CubicalTranslate(n_{PQ}, n_P)
$\lambda \leftarrow$ CubicalRatio$((x(Q) : 1), \widetilde{Q'})$ / CubicalRatio$((1 : 0), \widetilde{O})$
return $\lambda^{(p^2-1)/2^e}$

CHAPTER F

Quaternion algorithms (implementation details)*

This appendix records the quaternion algorithms that are called by the implementation. The algebraic background is given in Sections 3.2.2 and 3.2.3; here we keep only the implementation interface and the concrete subroutine dependencies. It is organized into three parts. The first part describes the lattice layer, including the data representation of quaternion elements and lattices, the normalization conventions used by the implementation, and the basic operations on lattices. The second part describes the ideal layer built on top of these lattice routines, including the representation of left ideals, order computations, and ideal constructions used by the higher-level quaternion algorithms.

F.1 Lattice algorithms

In the QIMEN-PRISM implementation, a quaternion algebra element α is stored as a pair $(\boldsymbol{a}, e)$, where $\boldsymbol{a} = (a_0, a_1, a_2, a_3)^T \in \mathbb{Z}^4$ and $e \in \mathbb{Z} \setminus \{0\}$. This represents

$$\alpha = \frac{a_0 + a_1\mathbf{i} + a_2\mathbf{j} + a_3\mathbf{k}}{e}$$

with respect to the fixed quaternion basis $(1, \mathbf{i}, \mathbf{j}, \mathbf{k})$.

A lattice L is stored as a pair (B, d), where $d \in \mathbb{Z} \setminus \{0\}$ and

$$B = (\boldsymbol{b}_0\ \boldsymbol{b}_1\ \boldsymbol{b}_2\ \boldsymbol{b}_3) = (b_{ir})_{0 \le i,r \le 3} \in M_4(\mathbb{Z}).$$

Here $\boldsymbol{b}_r = (b_{0r}, b_{1r}, b_{2r}, b_{3r})^T$ is the r-th column of B, so $\boldsymbol{b}_r/d$ is the coordinate vector of the r-th lattice basis element. Before lattice operations, the implementation puts the integer basis matrix B in HNF. After constructing a new lattice, it divides B and d by their common content, namely the gcd of all entries of B together with d, so that the stored denominator is reduced.

Algorithm 66 HNF(M)

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

for i from d down to 1 **do**
  for j from $i-1$ down to 1 **do**
    if $M_{i,j}$ and $M_{i,i}$ are both 0 **then**
      $g, u, v \leftarrow 1, 1, 0$
    else
      $g, u, v \leftarrow \mathsf{XGCD}(M_{i,i}, M_{i,j})$
    $M_i^\ell \leftarrow uM_i^\ell + vM_j^\ell$
    for j from $i-1$ down to 1 **do**
      $g \leftarrow M_{i,j}/M_{i,i}$
      $M_j^\ell \leftarrow M_j^\ell - gM_i^\ell$
    for j from $i+1$ up to c **do**
      $r \leftarrow M_{i,j} \bmod M_{i,i}$
      $g \leftarrow (M_{i,j} - r)/M_{i,i}$
      $M_j^\ell \leftarrow M_j^\ell - gM_i^\ell$
return M

Algorithm 67 LatticeEquality(L_1, L_2)

Input: Lattices L_1 and L_2 in HNF.
Output: true if $L_1 = L_2$, and false otherwise.

return $(|d_2|B_1 = |d_1|B_2)$

Algorithm 68 LatticeSum(L_1, L_2)

Input: Lattices L_1 and L_2 in HNF.
Output: $L_1 + L_2$ in HNF.

$M \leftarrow [\, d_1 B_2 \mid d_2 B_1 \,]$
$B \leftarrow \mathrm{HNF}(M)$
$d \leftarrow d_1 d_2$
Divide B and d by their common content.
return (B, d)

Algorithm 69 LatticeDual(L)

Input: A lattice $L = (B, d)$.
Output: The dual lattice L^*.

$M \leftarrow B^T$
$\Delta \leftarrow \det(M)$
$B' \leftarrow d \cdot \Delta \cdot M^{-1}$
$d' \leftarrow \Delta$
return (B', d') ▷ without HNF

Algorithm 70 LatticeIntersection(L_1, L_2)

Input: Lattices L_1 and L_2 in HNF.
Output: $L_1 \cap L_2$ in HNF.

1: $D_1 \leftarrow \text{LatticeDual}(L_1)$ and $D_2 \leftarrow \text{LatticeDual}(L_2)$
2: $S \leftarrow \text{LatticeSum}(D_1, D_2)$
3: $L \leftarrow \text{LatticeDual}(S)$
4: $L \leftarrow \text{HNF}(L)$
5: Reduce the denominator of L.
6: **return** L

Algorithm 71 LatticeMultiplication(L_1, L_2)

Input: Two lattices L_1 and L_2.
Output: $L_1 L_2$ in HNF.

1: $((\boldsymbol{b}_0, \boldsymbol{b}_1, \boldsymbol{b}_2, \boldsymbol{b}_3), d_1) \leftarrow L_1$
2: $((\boldsymbol{b}'_0, \boldsymbol{b}'_1, \boldsymbol{b}'_2, \boldsymbol{b}'_3), d_2) \leftarrow L_2$
3: $d \leftarrow d_1 d_2$
4: $S_0 \leftarrow (\boldsymbol{b}_0\boldsymbol{b}'_0, \ldots, \boldsymbol{b}_0\boldsymbol{b}'_3, \boldsymbol{b}_1\boldsymbol{b}'_0, \ldots, \boldsymbol{b}_1\boldsymbol{b}'_3)$
5: $H_0 \leftarrow \text{HNF}(S_0)$
6: $S_1 \leftarrow (\boldsymbol{b}_2\boldsymbol{b}'_0, \ldots, \boldsymbol{b}_2\boldsymbol{b}'_3, \boldsymbol{b}_3\boldsymbol{b}'_0, \ldots, \boldsymbol{b}_3\boldsymbol{b}'_3)$
7: $H_1 \leftarrow \text{HNF}(S_1)$
8: $B \leftarrow \text{HNF}([\, H_0 \mid H_1 \,])$
9: Divide B and d by their common content.
10: **return** (B, d)

Algorithm 72 LatticeContainment(α, L)

Input: A lattice L in HNF and a quaternion element $\alpha = \boldsymbol{a}/e$.
Output: A pair $(\mathsf{res}, \boldsymbol{c})$ where res states whether $\alpha \in L$, and $\boldsymbol{c}$ gives coordinates if membership holds.

1: $(B, d) \leftarrow L$
2: $\boldsymbol{w} \leftarrow d\boldsymbol{a}$
3: **for** $j = 3, 2, 1, 0$ **do**
4: **if** eb_{jj} does not divide w_j **then**
5: **return** (false, _)
6: $c_j \leftarrow w_j/(eb_{jj})$
7: $\boldsymbol{w} \leftarrow \boldsymbol{w} - c_j e\boldsymbol{b}_j$
8: **if** $\boldsymbol{w} = 0$ **then**
9: **return** $(\mathsf{true}, \boldsymbol{c})$
10: **else**
11: **return** (false, _)

Algorithm 73 LatticeInclusion(L_1, L_2)

Input: Two lattices L_1, L_2
Output: A boolean value: **true** if $L_1 \subset L_2$, **false** otherwise

1: **return** LatticeEquality(LatticeSum(L_1, L_2),L_2)

Algorithm 74 LatticeIndex(L_1, L_2)

Input: HNF lattices $L_1 \subseteq L_2$.
Output: The index $[L_2 : L_1]$.

$(B, d) \leftarrow L_1$
$(B', d') \leftarrow L_2$
for $i = 0, 1, 2, 3$ **do**
    $n \leftarrow d'^4 \prod_{i=0}^{3} b_{ii}$
    $m \leftarrow d^4 \prod_{i=0}^{3} b'_{ii}$
return $|n/m|$

Lattice reduction. Originating from prior work [NS09], the L2$_{\eta,\delta}$ algorithm receives an arbitrary lattice basis and its accompanying Gram matrix as initial inputs and returns an (η, δ)-reduced basis spanning the original lattice space, alongside the updated Gram matrix corresponding to this newly transformed basis. All high-level manipulations conducted on lattice basis elements and Gram matrix entries throughout the algorithm's outer loop framework are implemented via multi-precision integer arithmetic. By contrast, the internal intermediate computations rely on floating-point arithmetic to evaluate, store, and update the Gram-Schmidt orthogonalization (GSO) coefficients $r_{i,j}$ and $\mu_{i,j}$ for indices satisfying $i \geq j$. In [NS09], this collection of orthogonalization parameters is called the GSO family; further background is given in Chapter 5 of [NV10]. In the pseudocode below, all numerical scalars, vectors, and matrices stored in floating-point format are marked with the type annotation :: FLOAT, and every explicit conversion from integer data to floating-point data is denoted by FLOAT().

All vectors in the considered quadratic space have integral coordinates, and lattice bases are represented as integer matrices. The ExtendGSOFamily subroutine incrementally constructs Gram-Schmidt orthogonalization coefficients from the Gram matrix and the incomplete GSO family. The SizeReduce subroutine refines the basis vector $\boldsymbol{b}_k$, updating the GSO family and the Gram matrix whenever a projection coefficient exceeds the bound $\bar{\eta}$. The InsertBefore subroutine moves $\boldsymbol{b}_k$ before $\boldsymbol{b}_s$ in the basis order, then updates the inherited GSO coefficients and the corresponding diagonal orthogonalization value.

Algorithm 75 ExtendGSOFamily$(\boldsymbol{G}, k, r :: \text{Float}, \mu :: \text{Float})$

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

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

Algorithm 76 SizeReduce$((\boldsymbol{b}_0, \ldots, \boldsymbol{b}_{d-1}), \boldsymbol{G}, k, r :: \text{Float}, \mu :: \text{Float}, \bar{\eta} :: \text{Float})$

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

Output: $(\boldsymbol{b}_0, \ldots, \boldsymbol{b}_{d-1})$ size-reduced basis of the same lattice, its Gram matrix $\boldsymbol{G}$, the GSO family r, μ up to row k.

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

Algorithm 77 InsertBefore$((\boldsymbol{b}_0, \ldots, \boldsymbol{b}_{d-1}), \boldsymbol{G}, k, s :: \text{Float}, \mu :: \text{Float})$

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

Output: The basis $(\boldsymbol{b}_0, \ldots, \boldsymbol{b}_k, \boldsymbol{b}_s, \ldots, \boldsymbol{b}_{d-1})$ where $\boldsymbol{b}_k$ has been inserted before $\boldsymbol{b}_s$, the associated Gram matrix, the GSO family r, μ up to row s.

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

Algorithm 79 IdealGenerator(I)

Input: A left $\mathcal{O}_0$-ideal I.
Output: α primitive in I such that $I = \mathcal{O}(\alpha, \mathrm{nrd}(I))$.

$N_I \leftarrow \mathrm{nrd}(I)$
$n \leftarrow 0$
while true **do**
  $n \leftarrow n + 1$
  for a from $-n$ up to n **do**
    for b from $-n + |a|$ up to $n - |a|$ **do**
      for c from $-n + |a| + |b|$ up to $n - |a| - |b|$ **do**
        $d \leftarrow n - |a| - |b| - |c|$
        if $\gcd(a, b, c, d) = 1$ **then**
          $(B, \_, \_, \_) \leftarrow I$
          $\alpha \leftarrow a\boldsymbol{b}_0 + b\boldsymbol{b}_1 + c\boldsymbol{b}_2 + d\boldsymbol{b}_3$
          $q \leftarrow \mathrm{nrd}(\alpha)/N_I$
          if $\gcd(q, N_I) = 1$ **then return** α

Algorithm 78 L2$_{\eta,\delta}((\boldsymbol{b}_0, \ldots, \boldsymbol{b}_{d-1}), \boldsymbol{G})$

Input: A basis $(\boldsymbol{b}_0, \ldots, \boldsymbol{b}_{d-1})$ of a d-dimensional quadratic space, the associated $d \times d$ Gram matrix $\boldsymbol{G}$.
Output: A (η, δ)-reduced basis of the same lattice, its associated Gram matrix.

$\bar{\delta} \leftarrow \textsc{Float}\left(\frac{\delta+1}{2}\right)$, $\bar{\eta} \leftarrow \textsc{Float}\left(\frac{\eta+0.5}{2}\right)$
$r_{0,0} \leftarrow \textsc{Float}(\boldsymbol{G}_{0,0})$, $\mu_{0,0} \leftarrow \textsc{Float}(1)$,
$\boldsymbol{T} \leftarrow [\textsc{Float}(0), \textsc{Float}(0), \textsc{Float}(0), \textsc{Float}(0)]$
$k = 1$
while $k < d$ **do**
  $(\boldsymbol{b}_0, \ldots, \boldsymbol{b}_{d-1}), \boldsymbol{G}, r, \mu \leftarrow \textsc{SizeReduce}((\boldsymbol{b}_0, \ldots, \boldsymbol{b}_{d-1}), \boldsymbol{G}, k, r, \mu, \bar{\eta})$
  $T_0 \leftarrow \textsc{Float}(\boldsymbol{G}_{k,k})$
  for i from 1 up to $k - 1$ **do**
    $T_i \leftarrow T_{i-1} - \mu_{k,(i-1)} r_{k,(i-1)}$
  $s \leftarrow \min\{0 \le i \le k \mid \text{such that } T_j < \bar{\delta} r_{j,j} \text{ for all } i \le j < k\}$
  if $k \neq s$ **then**
    $(\boldsymbol{b}_0, \ldots, \boldsymbol{b}_{d-1}), \boldsymbol{G}, r, \mu \leftarrow \textsc{InsertBefore}((\boldsymbol{b}_0, \ldots, \boldsymbol{b}_{d-1}), \boldsymbol{G}, k, s, r, \mu)$
    $k \leftarrow s$
  $k \leftarrow k + 1$
return $(\boldsymbol{b}_0, \ldots, \boldsymbol{b}_{d-1}), \boldsymbol{G}$

F.2 Ideal algorithms

In the QIMEN-PRISM implementation, the data structure for storing an ideal relies on that of a lattice. The storage of a left $\mathcal{O}$-ideal I consists of the following information: a lattice L that represents I, and the reduced norm $\mathrm{nrd}(I)$ of I and its parent maximal order $\mathcal{O}$ which is stored as a lattice. In the following algorithms, an ideal I refers to $(L_I, \mathrm{nrd}(I), \mathcal{O}_I)$ or $(B_I, d_I, \mathrm{nrd}(I), B_{\mathcal{O}}, d_{\mathcal{O}})$.

Algorithm 80 IdealSum(I_1, I_2)

Input: Two left $\mathcal{O}$-ideals I_1, I_2.
Output: Left $\mathcal{O}$-ideal $I_1 + I_2$.

1: $L \leftarrow$ LatticeSum(L_1, L_2)
2: $N \leftarrow$ LatticeIndex$(L, \mathcal{O})$
3: $N \leftarrow \sqrt{N}$
4: **return** $(L, N, \mathcal{O})$

Algorithm 81 IdealMul(I, J)

Input: Two ideals I and J such that $\mathcal{O}_R(I) = \mathcal{O}_L(J)$.
Output: IJ.

1: $N \leftarrow \mathrm{nrd}(I)\,\mathrm{nrd}(J)$
2: $(L_1, \_, \mathcal{O}) \leftarrow I$
3: $(L_2, \_, \_) \leftarrow J$
4: **return** (LatticeMultiplication$(L_1, L_2), N, \mathcal{O})$

Algorithm 82 IdealIntersection(I_1, I_2)

Input: Two left $\mathcal{O}$-ideals I_1 and I_2.
Output: Left $\mathcal{O}$-ideal $I_1 \cap I_2$.

1: $L \leftarrow$ LatticeIntersection(L_1, L_2)
2: $N \leftarrow$ LatticeIndex$(L, \mathcal{O})$
3: $N \leftarrow \sqrt{N}$
4: **return** $(L, N, \mathcal{O})$

Algorithm 83 IdealInverse(I)

Input: A left $\mathcal{O}$-ideal I.
Output: The inverse of I (stored as lattice)

1: $(B, d, N, \_) \leftarrow I$
2: $C \leftarrow \mathrm{diag}(1, -1, -1, -1)$
3: $B' \leftarrow CB$
4: $d' \leftarrow d \cdot N$
5: **return** (B', d')

Algorithm 84 RightOrderOfAnIdeal(I)

Input: A left $\mathcal{O}$-ideal I.
Output: Right order of I.

1: $(L, \_, \_) \leftarrow I$
2: **return** LatticeMultiplication(IdealInverse$(I), L)$

Algorithm 85 IdealCreatePrincipal$(x, \mathcal{O})$

Input: An element $x = \boldsymbol{a}/d_x$ and an order $\mathcal{O} = (B_{\mathcal{O}}, d_{\mathcal{O}})$.
Output: Left-$\mathcal{O}$ ideal $\mathcal{O}x$.

$((\boldsymbol{b}_0, \boldsymbol{b}_1, \boldsymbol{b}_2, \boldsymbol{b}_3), d_{\mathcal{O}}) \leftarrow \mathcal{O}$
for $i = 0, 1, 2, 3$ **do**
    $\boldsymbol{m}_i \leftarrow \boldsymbol{b}_i \cdot x$
$B \leftarrow (\boldsymbol{m}_0, \boldsymbol{m}_1, \boldsymbol{m}_2, \boldsymbol{m}_3)$ ▷ The columns of B are the coordinates of $\boldsymbol{b}_i x$
$d \leftarrow d_x \cdot d_{\mathcal{O}}$
Divide B and d by their common content.
$B \leftarrow \mathrm{HNF}(B)$
$N \leftarrow \mathrm{nrd}(x)$
return $(B, d, N, \mathcal{O})$

Algorithm 86 IdealCreatePrimitive$(x, \mathcal{O}, N)$

Input: A primitive element $x \in \mathcal{O}$ and an integer N.
Output: $\mathcal{O}x + \mathcal{O}N$

$I \leftarrow$ IdealCreatePrincipal$(x, \mathcal{O})$
$B \leftarrow N \cdot B_{\mathcal{O}}$ ▷ Matrix by integer multiplication
$d \leftarrow d_{\mathcal{O}}$
$(L_1, \_, \_) \leftarrow I$
$L_2 \leftarrow (B, d)$
$L \leftarrow$ LatticeSum(L_1, L_2)
$N' \leftarrow$ LatticeIndex$(L, \mathcal{O})$
$N' \leftarrow \sqrt{N'}$
return $(L, N', \mathcal{O})$

Algorithm 87 ConnectingIdeal$(\mathcal{O}_L, \mathcal{O}_R)$

Input: Two orders $\mathcal{O}_L$ and $\mathcal{O}_R$.
Output: $N\mathcal{O}_L\mathcal{O}_R$, where N is the square root of the index of $\mathcal{O}_L \cap \mathcal{O}_R$ in $\mathcal{O}_L$.

$I \leftarrow$ LatticeIntersection$(\mathcal{O}_L, \mathcal{O}_R)$
$N \leftarrow$ LatticeIndex$(I, \mathcal{O}_L)$
$N \leftarrow \sqrt{N}$
$(B, d) \leftarrow$ LatticeMultiplication$(\mathcal{O}_L, \mathcal{O}_R)$
$B \leftarrow N \cdot B$
Divide B and d by their common content
return (B, d)