QIMEN-PIKE

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 key-encapsulation mechanism QIMEN-PIKE.

1.1 Design rationale

QIMEN-PIKE is based on the quantum-safe isogeny-based ElGamal-type public-key encryption scheme PIKE [CCLL26](accepted at PKC'26), which itself builds on POKÉ [BM25]. It follows the same mathematical hardness assumption and protocol architecture.

Mathematical problem. The core hard problem underlying QIMEN-PIKE is a masked variant of the supersingular isogeny problem with torsion-point information. Let E and E' be supersingular elliptic curves, let $\phi : E \to E'$ be a secret isogeny whose restriction to $E[N]$ is an isomorphism, and let (P, Q) be a basis of $E[N]$. For a prescribed masking group $\Gamma \subseteq \mathrm{GL}_2(\mathbb{Z}/N\mathbb{Z})$, the masked torsion isogeny problem is, given

$$E, \qquad E', \qquad (P, Q), \qquad M \begin{pmatrix} \phi(P) \\ \phi(Q) \end{pmatrix}$$

for an unknown matrix $M \in \Gamma$, to recover ϕ. The unmasked case $M = I_2$ is the supersingular isogeny problem with torsion-point information underlying SIDH and SIKE [JD11, JAC+22].

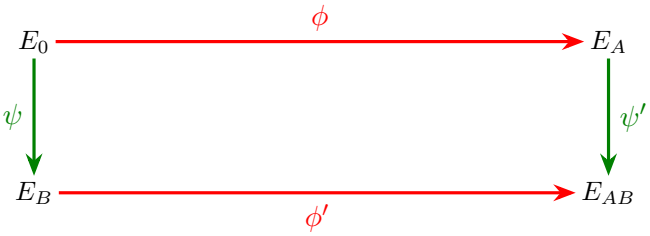

Figure 1.1: Design rationale of isogeny-based ElGamal-type public key encryption. The receiver's secret isogeny ϕ defines the public key E_A. With sufficient auxiliary information attached to E_A, the sender can derive a compatible green isogeny path ψ' and obtain E_{AB} as the target curve for encrypted delivery.

The *unmasked* torsion images published in SIDH/SIKE enabled polynomial-time key-recovery attacks [CD23, MMP+23, Rob23]. These attacks rely essentially on the algebraic relations exposed by $\phi(P)$ and $\phi(Q)$, and therefore do not directly apply when the torsion images are hidden by an unknown masking matrix M. Since the fall of SIDH/SIKE, the masked isogeny problem has been proposed as a core hardness assumption in isogeny-based cryptography, and has been used in the design of numerous protocols such as M-SIDH [FMP23], FESTA [BMP23], QFESTA [NO24], LIT-SiGamal [MS25], among others. Cryptanalysis work has also been done towards this variant of the isogeny problem including [CV23] and [DFP24]. However, for the masked-torsion instances used in QIMEN-PIKE and aforementioned schemes, no polynomial-time recovery algorithm is currently known, and the best known applicable algorithms remain of exponential complexity.

Core design idea. QIMEN-PIKE follows the isogeny-based ElGamal-type architecture illustrated in Fig. 1.1. The receiver samples a secret isogeny $\phi : E_0 \to E_A$ and publishes E_A together with masked images of several torsion points under ϕ. The sender then samples an ephemeral smooth-degree isogeny $\psi : E_0 \to E_B$ and uses the published torsion information to construct its pushforward $\psi' : E_A \to E_{AB}$. This yields a commutative diagram satisfying $\phi' \circ \psi = \psi' \circ \phi$, while the masking prevents the sender from recovering the receiver's secret isogeny.

Once the commutative diagram is established, QIMEN-PIKE derives the shared secret from a pairing value. The sender transmits one masked D-torsion point on E_B and one on E_{AB}. Using the secret masking information, the receiver evaluates a single two-dimensional isogeny to obtain the corresponding points on an intermediate curve and computes their pairing. By the compatibility of pairings with isogenies and the commutativity of the diagram, this recovers the same pairing-derived value computed by the sender.

FO Transform. Following the standard Fujisaki-Okamoto transform [FO99], the IND-CPA-secure public-key encryption scheme described above is transformed into an IND-CCA2-secure key-encapsulation mechanism. The encapsulation algorithm samples a ran-

dom message and hashes it together with the public key to derive both the key material and deterministic encryption randomness, before encrypting the message using QIMEN-PIKE.PKE. During decapsulation, the receiver decrypts the ciphertext and re-encrypts the recovered message to validate it. A valid ciphertext yields the same shared secret, while an invalid ciphertext is handled through implicit rejection [Per12] using a secret fallback value. The IND-CCA2 security of the resulting KEM follows from the IND-CPA security of QIMEN-PIKE.PKE in the random oracle model.

Parameter selection and failure analysis. The three parameter sets of QIMEN-PIKE, 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 PIKE

While building on the core design of PIKE [CCLL26], QIMEN-PIKE introduces several theoretical and engineering innovations that distinguish it from prior academic prototypes. These differences are summarized as follows.

- **New parameters and optimized procedures.** We instantiate QIMEN-PIKE with a new set of security parameters specifically tailored to meet the diverse security-level requirements of the NGCC submission. Furthermore, we deploy targeted elliptic curve arithmetic optimizations, including accelerated scalar multiplications and an improved three-point ladder algorithm tuned to these chosen parameters. These low-level algorithmic refinements significantly reduce the total number of underlying field operations, ensuring that QIMEN-PIKE not only meets the rigorous NGCC security standards but also achieves state-of-the-art computational efficiency across all targeted categories.
- **Tailored public-key/ciphertext compression.** We propose two optimization techniques to optimize the sizes of the public key and ciphertext in QIMEN-PIKE. The first technique mitigates storage requirements via point combination. The second technique achieves strictly higher compression by encoding the points as scalar representations with respect to fixed torsion bases, thereby trading computational efficiency for a reduced representation size. Accordingly, we develop two distinct implementations: an uncompressed variant and a compressed variant. Although the compressed implementation is less efficient than the uncompressed one, it yields a significantly more compact public key (by $\approx 37\%$) and ciphertext (by $\approx 25\%$).
- **Assembly-level field arithmetic kernels.** Our implementation incorporates hand-optimized assembly routines that leverage x86-64 BMI2 and ADX instructions. By

refining the unsaturated-radix limb representation and minimizing carry-propagation overhead during modular multiplication, the cryptographic suite achieves runtime reductions of over 10% compared to generic multiprecision baselines.

1.2 Feature Statements

1.2.1 Innovativeness

Most existing isogeny-based public-key encryption schemes developed after the attacks on SIDH rely on higher-dimensional isogenies and auxiliary torsion-point information, as exemplified by [BMP23, NO24, BM25, MS25]. Among these constructions, POKÉ [BM25] forms the principal basis of PIKE [CCLL26] and QIMEN-PIKE, deriving the shared secret from the images of a torsion basis under the shared isogeny.

In contrast, PIKE, and consequently QIMEN-PIKE, follow a distinct pairing-assisted encryption architecture. The principal innovations lie in the specific shared-secret relation and decryption mechanism introduced within this paradigm. In the following, we refer to QIMEN-PIKE throughout, while noting that these protocol-level innovations originate from PIKE.

- **Pairing-derived shared-secret relation:** QIMEN-PIKE is the first isogeny-based public-key encryption scheme in which the shared secret is derived from a pairing value rather than from the j-invariant of an elliptic curve or the images of a full torsion basis [BMP23, NO24, BM25, MS25]. The sender evaluates a pairing on D-torsion points related through the commutative isogeny diagram and uses the resulting value to mask the message. The mechanism accelerates the decryption by a factor of 1.24.
- **Relaxed smoothness requirement on rational torsion using pairings.** To the best of our knowledge, QIMEN-PIKE is the first isogeny-based public-key encryption scheme to use pairings to relax the smoothness requirement on the order D of the $\mathbb{F}_{p^2}$-rational torsion used for shared-secret recovery. Since the receiver recovers the shared value directly from a pairing, decryption does not require solving a discrete logarithm in a group of order D. Consequently, D need not be smooth, relaxing the required form of the field characteristic and allowing a smaller prime p to be selected. The shorter prime accelerates the keygen, encapsulation, decapsulation by another factors from 1.24 to 1.3.

Together, these features establish QIMEN-PIKE as an independent pairing-assisted isogeny-based encryption architecture rather than a known modification or parameterization of prior works.

1.2.2 Simplicity

A central design advantage of QIMEN-PIKE is the simplicity of its decryption pathway. Earlier higher-dimensional isogeny-based PKEs in the POKÉ family [BM25] derive the shared secret from torsion-point images, which forces the receiver to reconstruct a commutative diagram through an application of Kani's Lemma and to resolve a smooth discrete logarithm in order to recover the message. QIMEN-PIKE dispenses with this machinery entirely: by transporting the shared secret through *pairing values* rather than basis images, decryption reduces to evaluating a small number of pairings and a single inversion, with no Kani gluing step and no smooth discrete-logarithm computation on the receiver side. The protocol retains the clean ElGamal-type structure of a single secret isogeny defining the public key and a sender-side isogeny producing the ciphertext (see Figure 1.1), which keeps both the specification and the security argument compact. The same simplification also relaxes the arithmetic constraints on the underlying prime, so that the field can be chosen smaller and the implementation kept correspondingly lean.

1.2.3 Flexibility and Compatibility

The modular structure of QIMEN-PIKE provides flexibility across diverse engineering environments and compatibility with standard KEM transforms:

- **Standard KEM transform.** QIMEN-PIKE.KEM is obtained from the underlying public-key encryption scheme QIMEN-PIKE.PKE through a standard Fujisaki-Okamoto-type transform, yielding IND-CCA2 security in the (quantum) random oracle model from the IND-CPA security of the PKE. The transform is applied as a self-contained wrapper, so the core encryption and decryption routines remain unchanged across security levels.
- **Extensible hash/XOF backends.** The protocol separates the high-level hashing and key-derivation routines from the underlying extendable output function (XOF). For standard global deployments it natively supports a SHAKE256 instantiation, while for compliance with national commercial cryptography standards it can be driven by an SM3-based construction. This allows the same protocol machinery to be compiled for different regulatory domains via simple build-time switches.
- **Flexible point compression modes.** The representation of the public key and ciphertext can be tailored to the bandwidth and computational profile of the target platform. QIMEN-PIKE supports an uncompressed, coordinate-centric mode (ideal for minimized computation) as well as a compressed, basis-coefficient mode that encodes points as scalars with respect to fixed torsion bases, trading a modest amount of computation for significantly smaller transmission sizes.

1.2.4 Extensibility

The ElGamal-type structure of QIMEN-PIKE makes it a convenient building block for larger protocols. Because the shared secret is transported through pairing values on a public commutative diagram rather than through scheme-specific auxiliary data, the same key-generation and encapsulation interfaces extend naturally to related functionalities:

- **Well-formedness proofs.** QIMEN-PIKE supports efficient proofs of well-formedness for both public keys and ciphertext [LM26]. These proofs allow a receiver or a sender to demonstrate that a public key or a ciphertext are valid without revealing the underlying information. This property facilitates the secure use of QIMEN-PIKE as a building block in more advanced cryptographic protocols, where malformed public keys or ciphertexts could otherwise invalidate the security analysis.
- **Public-key encryption and key encapsulation.** The same underlying construction yields both a public-key encryption scheme (QIMEN-PIKE.PKE) and a key-encapsulation mechanism (QIMEN-PIKE.KEM), so applications needing either primitive share a single, well-analyzed core.
- **Static and ephemeral deployments.** The clean separation between the receiver's long-term secret isogeny and the sender's per-message isogeny supports both static keys (for repeated encapsulation to a fixed recipient) and ephemeral keys (for forward-secret session establishment) without changing the protocol logic.
- **Composable compression layer.** The point-compression layer is decoupled from the protocol layer, so future compression techniques or alternative point encodings can be substituted without affecting correctness or the security reduction.

1.2.5 Performance

- **Compact public keys and ciphertexts.** QIMEN-PIKE is particularly attractive for applications in which public-key must be transmitted or stored under tight communication and memory constraints. Examples include constrained IoT devices communicating over low-rate networks, smart cards and contactless devices with limited transmission capacity, intermittently connected embedded systems, and satellite or remote telemetry links. In these settings, reducing public-key and ciphertext sizes helps decrease transmission time, packet fragmentation, storage requirements, and retransmission overhead. At NIST security level 5, the public key and ciphertext of QIMEN-PIKE with NGCC-2 require only 595 B and 882 B, respectively. As shown in Table 1.1, these are approximately 2.6× and 1.8× smaller than those of ML-KEM-1024 (Kyber) [SAB+20], and approximately 12.2× and 16.4× smaller than those of HQC-256 [AAB+22], respectively.
- **Reasonable encapsulation and decapsulation latency.** Although QIMEN-PIKE is slower than ML-KEM and HQC, its encapsulation and decapsulation latency remains within tens of milliseconds. For example, at the NGCC-1 security level, our

Table 1.1: Comparison of public-key and ciphertext sizes, in bytes, for NIST-standardized or selected-for-standardization post-quantum KEMs at security level NIST-5, which is the same as QIMEN-PIKE with NGCC-2.

Scheme	Public key	Ciphertext
QIMEN-PIKE (NGCC-2)	595 **B**	882 **B**
ML-KEM-1024 (Kyber)	1568 B (2.6×)	1568 B (1.8×)
HQC-256	7245 B (12.2×)	14 469 B (16.4×)

optimized 64-bit implementation requires approximately 18 ms for encapsulation and 29 ms for decapsulation on an Intel Ultra 9 CPU running at 2.7 GHz (see Table 7.4). These results show that the substantial reduction in communication size is achieved without incurring prohibitive online computation costs.

1.2.6 Diversity of assumptions

QIMEN-PIKE relies on isogeny-based assumptions, which are fundamentally different from those underlying lattice-based and code-based constructions. Therefore, QIMEN-PIKE 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-PIKE 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-PIKE 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 1.2: Notation and parameters of the QIMEN-PIKE.

	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 1.3: Notation and parameters of the QIMEN-PIKE.

	Security Parameters
λ_c	Target classical security strength, in bits.
λ_q	Target quantum security strength, in bits.
λ	Internal security parameter, set to $2\lambda_q$.
	Public Parameters
pp	Public parameters, including $p, a, C, C_1, C_2, D, E_0$, the torsion bases, and w_0.
p	Field characteristic.
a	Power-of-two torsion parameter appearing in $p+1 = 2^a C_1 D$.
C_1, C_2, C	Odd torsion parameters with $C = C_1 C_2$, $C_1 \mid p+1$ $C_2 \mid p-1$, and $\gcd(C_1, C_2) = 1$.
D	Torsion order used for the pairing-derived shared value.
(P_0, Q_0)	Basis of $E_0[2^a]$.
(R_0, S_0)	Basis of $E_0[C]$.
(X_0, Y_0)	Basis of $E_0[D]$.
w_0	Tate pairing value $t_D(X_0, Y_0)$.
	CPA-PKE Objects
Π_{CPAPKE}	IND-CPA public-key encryption scheme QIMEN-PIKE.PKE = $(\mathsf{QIMEN-PIKE.PKE.KeyGen}, \mathsf{QIMEN-PIKE.PKE.Encrypt}, \mathsf{QIMEN-PIKE.PKE.Decrypt})$.
$\mathsf{QIMEN-PIKE.PKE.KeyGen}$	Key-generation algorithm for QIMEN-PIKE.PKE.
$\mathsf{QIMEN-PIKE.PKE.Encrypt}$	Encryption algorithm for QIMEN-PIKE.PKE.
$\mathsf{QIMEN-PIKE.PKE.Decrypt}$	Decryption algorithm for QIMEN-PIKE.PKE.
q	Secret integer used to define the key-generation isogeny degree $q(2^{a-2} - q)$.
$\alpha_2, \beta_2, \delta_D$	Secret masking scalars stored in the PKE secret key.
γ_C	Masking scalar for the C-torsion images, sampled from $(\mathbb{Z}/C\mathbb{Z})^\times$; not stored in the secret key.
pk	Public key $(E_A, (P_A, Q_A), (R_A, S_A), X_A)$.
sk	Secret key $(q, \alpha_2, \beta_2, \delta_D)$ for QIMEN-PIKE.PKE.
	Encryption and Decryption Objects
r	Encryption nonce defining the degree-C isogenies through $R_0 + [r]S_0$ and $R_A + [r]S_A$.
ω_2	Masking scalar for the transmitted 2^a-torsion images.
η_D and ζ_D	Masking scalars for the transmitted D-torsion images.
E_B and E_{AB}	Codomain curves produced during encryption.
E_M	Intermediate curve used during decryption to recover the pairing value.
pad	Pad derived from $\mathsf{H}(\mathsf{KDF} \Vert \mathsf{Tr}(w'))$ and xored with the message.
ct	Ciphertext containing E_B, E_{AB}, the masked image points, and $\mathsf{pad} \oplus \mathsf{msg}$.
	CCA-KEM Objects
QIMEN-PIKE.KEM	CCA-secure KEM obtained from QIMEN-PIKE.PKE through a Fujisaki–Okamoto-style transform.
$\mathsf{Enc.Det}$	Deterministic encryption algorithm used in the KEM re-encryption check.
$\mathsf{H_{pk}}$	Hash of the public key.
$\mathsf{H_{ct}}$	Hash of the ciphertext transcript.
G	Hash-and-expand function producing $(\bar{K}, \rho)$.
KDF	Key-derivation function for the final shared secret.
ρ	Seed used to deterministically derive encapsulation coins.
z	Fallback secret stored in the KEM secret key for implicit rejection.
K	Final shared secret output by encapsulation and decapsulation.

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-PIKE 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-PIKE 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-PIKE. 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*.

2.2.1.1 Isomorphisms between Montgomery curves

For a Montgomery curve $E_{A,B}$, we define its j-invariant

$$j(E_{A,B}) = \frac{256(A^2 - 3)^3}{A^2 - 4}. \tag{2.5}$$

The j-invariant characterizes isomorphism classes of elliptic curves over the algebraic closure, i.e., two curves have the same j-invariant if and only if they are isomorphic or twists of one another. Given two curves $E_{A,B}$ and $E_{A',B'}$ that are isomorphic, the change of

coordinates $(x, y) \mapsto (Dx + R, Cy)$ that maps the former onto the latter is given by

$$D = \lambda^2 \frac{B'}{B}, \quad R = \lambda^2 \frac{AB'}{3B} - \frac{A'}{3}, \quad C = \lambda^3 \frac{B'}{B}, \quad \lambda := \sqrt{\frac{B(2A'^3 - 9A')(3 - A^2)}{B'(2A^3 - 9A)(3 - A'^2)}}. \tag{2.6}$$

The formulas are implemented by the IsomorphismMontgomeryCurves algorithm described in Chapter C.

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

$$x_R = B\lambda^2 - (x_P + x_Q) - A, \tag{2.7}$$

and

$$y_R = \lambda(x_P - x_R) - y_P, \tag{2.8}$$

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]:

$$r_\pm = \frac{B_{XZ} \pm \sqrt{B_{XZ}^2 - B_{ZZ}B_{XX}}}{B_{ZZ}}, \tag{2.9}$$

where

$$\begin{aligned} B_{XX} &= (x_P x_Q - 1)^2, \\ B_{XZ} &= (x_P x_Q + 1)(x_P + x_Q) + 2A x_P x_Q, \\ B_{ZZ} &= (x_P - x_Q)^2. \end{aligned} \tag{2.10}$$

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 (2.10). The resulting algorithm is given in RecoverCodomain, see Section C.2.

2.2.3 Torsion subgroups and deterministic basis computation

In QIMEN-PIKE, we precompute several torsion bases on the public curve E_0. As discussed in Section 3.1.1, we generate bases for $E_0[2^a]$, $E_0[C]$ and $E_0[D]$, respectively. Since they are precomputed, we do not discuss in detail how they are generated in this specification.

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 in the context of QIMEN-PIKE, 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-PIKE

In QIMEN-PIKE, pairings are used crucially in the design of the protocol to derive a shared secret between encryption and decryption. Specifically, for an integer D and a specified basis (X_0, Y_0) of $E_0[D]$, which are scheme parameters specified in Section 5.1, we define the value $w_0 := t_D(X_0, Y_0)$ as a protocol parameter pp, where t_D is the degree-D Tate pairing. Later, in decryption, we compute $w = t_D(-X_M, Y_M)$ for a similar basis (X_M, Y_M), and a specific relation between w_0 and w allows us to recover the same value w' as a power of either w_0 or w, which is the crucial shared secret required to encrypt and decrypt.

2.4 1-Dimensional isogenies

For two elliptic curves E_1 and E_2 over $\mathbb{F}_q$, an *isogeny* is a non-constant map $\varphi : E_1 \to E_2$ defined coordinate-wise by polynomial fractions over $\mathbb{F}_q$, that satisfies $\varphi(0_{E_1}) = 0_{E_2}$. 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 $\deg(\varphi) = \max\{\deg(f_1), \deg(f_2)\}$.

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

In QIMEN-PIKE, we compute ℓ-isogeny chains for $\ell \in \{3, 5, 7, 11, 13\}$ using the specialization of Vélu's formulas to Montgomery curves [CH17], which apply for an isogeny

$\phi : E_A \to E_{A'}$ of prime degree $\ell \neq 2$. In general, such isogenies can be written in the form

$$\phi : (x, y) \mapsto (f(x), c \cdot y \cdot f'(x)),$$

for some constant $c \in \mathbb{F}_{p^2}$ and a rational function $f(x)$ with derivative $f'(x)$. Let P be the generator of the kernel $\ker \phi$, then we can derive both c and $f(x)$ from P by

$$c := \prod_{0<s<\frac{\ell-1}{2}} x_{[s]P}, \qquad f(x) := x \cdot \prod_{0<s<\ell} \frac{x_{[s]P} \cdot x - 1}{x - x_{[s]P}}.$$

These formulas can be optimized for degree and x-only arithmetic, as we only need $x_{\phi(Q)}$ for $Q \in E_A$ for most applications in QIMEN-PIKE. We provide the details of the implementation of these isogenies in OddIsogenyChain.

2.5 2-Dimensional isogenies

In QIMEN-PIKE, 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

$$(P_1, P_2) + (Q_1, Q_2) := (P_1 + Q_1,\ P_2 + Q_2), \tag{2.11}$$

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

$$C : y^2 = f(x), \tag{2.12}$$

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 $O\left(\sum e_i \ell_i^2\right)$.

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

In QIMEN-PIKE. For QIMEN-PIKE, 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-PIKE, 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.

Remark 2.5.1. *In QIMEN-PIKE, in contrast with other isogeny-based schemes, we compute $(2,2)$-isogenies whose degree is almost equal to the available 2-torsion 2^a. We therefore denote by 2^a the available 2-torsion, and will mostly use $(2^{a-2}, 2^{a-2})$-isogenies. This is in contrast with other literature, where the notation would be to have degree $(2^a, 2^a)$ using points of order 2^{a+2}.*

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

In Chapter E, 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 E, 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-PIKE, 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]

2.5.3 Implementation details for chains of isogenies

In Chapter E, 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.

2.6 Quaternions

2.6.1 Big integers

QIMEN-PIKE 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-PIKE 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;

[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.

- 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-PIKE);
- 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-PIKE reference implementation.

Algorithm 3 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 $x^2 \equiv n \pmod{m}$

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

2.6.2 Basic definitions of quaternion algebra

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

$$\mathbf{i}^2 = -1, \qquad \mathbf{j}^2 = -p, \qquad \mathbf{ij} = -\mathbf{ji} = \mathbf{k}. \tag{2.13}$$

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. (2.13).

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

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-PIKE 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 $(1, \mathbf{i}, \mathbf{j}, \mathbf{k})$.

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 $\mathrm{End}(E)$. $\mathrm{End}(E)$ is isomorphic to a maximal order $\mathcal{O}$ in the quaternion algebra $\mathcal{B}_{p,\infty}$. Fixing an isomorphism $\mathcal{O} \simeq \mathrm{End}(E)$, an element $\alpha \in \mathcal{O}$ corresponds to an endomorphism of E, and we write, by slight abuse of notation, $\alpha(P)$ for $P \in E$ to denote the image of α under a (fixed) isomorphism evaluated at $P \in E$, and similarly, we write $\ker \alpha$ to denote the kernel of the image of α.

2.6.3 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 $p \equiv 3 \bmod 4$.

2.6.3.1 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 4 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-PIKE, 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-PIKE. 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 5 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 $(\mathrm{Re}(z), \mathrm{Im}(z))$

2.6.3.2 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

$$x^2 + y^2 + p(z^2 + t^2) = 4M \tag{2.14}$$

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. (2.14). The algorithm GeneralizedRepresentInteger proceeds as follows: it first samples z, t in an appropriate range as specified in the algorithm, then computes the value $M' := 4M - p(z^2 + t^2)$. It calls CornacchiaGeneral with the precomputed list `cornacchia_prime_list` to solve the binary quadratic equation $x^2 + y^2 = M'$. If this call returns $\perp$, the current candidate is discarded and the algorithm continues by sampling new values of z, t.

Algorithm 6 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 $M' > 0$ **then**
        $(x, y) \leftarrow$ CornacchiaGeneral(M', `cornacchia_prime_list`)
        if $(x, y) \neq \perp$ **then**
            `found` $\leftarrow$ `true`
            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`")

CHAPTER 3

Protocol

3.1 Protocol parameters

This section specifies the protocol parameters for QIMEN-PIKE, our new **P**airing-assisted, **I**sogeny-based **K**ey **E**ncryption scheme. We refer the reader to Chapter 4 for a description of how to efficiently represent the exchanged data.

As in POKÉ [BM25], QIMEN-PIKE follows the ElGamal paradigm: encryption derives a shared secret and combines it with the message to form the ciphertext. A protocol overview is given in Figure 3.1. The full specifications of QIMEN-PIKE.PKE and QIMEN-PIKE.KEM are given in the subsequent sections.

3.1.1 Parameter requirements

The QIMEN-PIKE algorithm works with several parameters and precomputed values. All algorithms implicitly take a parameter set as input. For specific values of such parameters, we refer the reader to Chapter 5. Hereafter we provide a list of these parameters, and in some cases also describe how these can be computed from other parameters.

3.1.2 Scheme parameters

The parameters and precomputed values include:

- The internal security parameter λ, set to $2\lambda_q$.
- The positive integers a, C_1, C_2, D such that $p+1 = 2^a C_1 D$, $C_2 \mid p-1$, and $\gcd(C_1, C_2) = 1$. Their exact values are defined in Section 5.1. Additionally, we fix $C = C_1 C_2$.
- E_0 is the curve with equation $y^2 = x^3 + x$.
- (P_0, Q_0) is a basis of $E_0[2^a]$.
- (R_0, S_0) is a basis of $E_0[C]$.
- (X_0, Y_0) is a basis of $E_0[D]$.
- w_0 denotes the Tate pairing value $t_D(X_0, Y_0)$.

- $\mathsf{H} : \{0,1\}^\star \to \{0,1\}^{2\lambda}$ is a hash function. For domain separation, key-derivation queries shall be written as $\mathsf{H}(\mathsf{KDF} \mid \cdot)$.

3.1.3 Protocol Sketch

We conclude this section with a sketch of the protocol to help with the visualization of some of the parameters defined above.

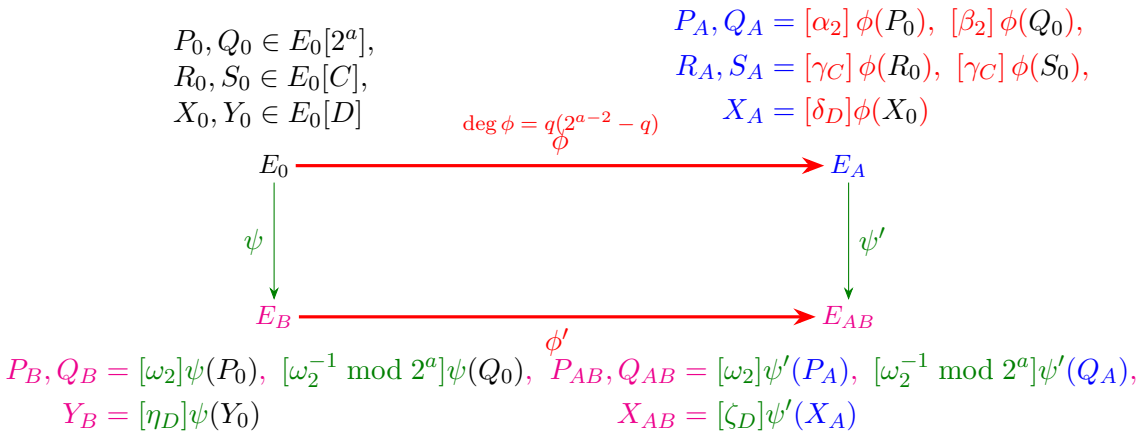

Figure 3.1: A sketch of QIMEN-PIKE. Black: public parameters; red: secret key; blue: public key; dark green: encryption nonce; magenta: ciphertext.

3.2 Specification of PIKE.PKE

This section presents the IND-CPA secure public-key encryption scheme QIMEN-PIKE.PKE as a collection of three algorithms QIMEN-PIKE.PKE.KeyGen,QIMEN-PIKE.PKE.Encrypt, and QIMEN-PIKE.PKE.Decrypt.

3.2.1 Key Generation

We split the key generation algorithm in two subsections. In Section 3.2.1.2, we describe the subroutines, which we then use to introduce the key generation algorithm in Section 3.2.1.3.

3.2.1.1 Precomputed data

The key-generation subroutines use the fixed isomorphism $\mathcal{O}_0 \simeq \mathrm{End}(E_0)$ described in Section 2.6.2. Recall that

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

with integral basis $\left(1, \mathbf{i}, \frac{\mathbf{i}+\mathbf{j}}{2}, \frac{1+\mathbf{k}}{2}\right)$. For QIMEN-PIKE, the following data are precomputed:

- a basis (R_0, S_0) of $E_0[C]$, together with matrices $M_1, \ldots, M_4 \in \mathsf{M}_2(\mathbb{Z}/C\mathbb{Z})$ representing the actions of $1, \mathbf{i}, \frac{\mathbf{i}+\mathbf{j}}{2}, \frac{1+\mathbf{k}}{2}$ on $E_0[C]$ with respect to (R_0, S_0).

These matrices allow EndomorphismToKernel to evaluate the action of any endomorphism $\alpha \in \mathcal{O}_0$ on $E_0[C]$ by reducing the coordinates of α in the above integral basis modulo C.

3.2.1.2 Subroutines

This subsection specifies the auxiliary procedures required by key generation. These procedures define how the isogeny used in key generation is computed and how its evaluations are obtained. They do not, by themselves, define the secret key or the public key.

The main procedure, QIMEN-PIKE.CoreKeyGenIso, first computes an isogeny $\varphi : E_0 \to E_A$ of degree $q(2^{a-2} - q)$, together with its evaluations on the basis of $E_0[2^a]$ and on the designated point of $E_0[CD]$. This procedure follows the heuristic non-smooth isogeny $\varphi : E_0 \to E_A$ generation method of POKÉ [BM25], with implementation-level optimizations. It first samples an endomorphism θ of degree $q(2^{a-2} - q)C$ via GeneralizedRepresentInteger. Then, it extracts a component of degree $q(2^{a-2} - q)$: this is achieved by computing the isogeny $\psi : E_0 \to E_A$ with kernel $\ker(\bar{\theta}) \cap E_0[C]$ via EndomorphismToKernel and then defining φ to be equal to $[1/C] \circ \psi \circ \theta$.

EndomorphismToKernel We detail the process of extracting the kernel of a given endomorphism $\alpha \in \mathrm{End}(E_0)$ restricted to the C-torsion in EndomorphismToKernel. Given an endomorphism α with $C \mid \deg(\alpha)$, the algorithm works by evaluating its action on the fixed C-torsion basis (R_0, S_0) of $E_0[C]$.

By representing α over the fixed integral basis of $\mathcal{O}_0$, we have

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

Using the precomputed matrices $M_1, \ldots, M_4 \in \mathsf{M}_2(\mathbb{Z}/C\mathbb{Z})$ from Section 3.2.1.1, one can explicitly recover the action matrix $\mathbf{M}_\alpha \in \mathsf{M}_2(\mathbb{Z}/C\mathbb{Z})$ as

$$\mathbf{M}_\alpha = x_1M_1 + x_2M_2 + x_3M_3 + x_4M_4.$$

We then compute a vector $[v_1, v_2]^T \in \ker \mathbf{M}_\alpha$, which directly gives the kernel generator $K = [v_1]R_0 + [v_2]S_0$ for the corresponding degree-C isogeny.

3.2.1.3 Key Generation

The key-generation procedure as specified in QIMEN-PIKE.PKE.KeyGen samples the receiver secret data, invokes the subroutines above to compute the required isogeny images, applies the public-key masking, and outputs the key pair. More precisely:

Algorithm 7 QIMEN-PIKE.CoreKeyGenIso(λ)

Input: The internal security parameter λ

Output: $(q, E_A, (P_A, Q_A), (R_A, S_A), X_A)$, where $\varphi : E_0 \to E_A$ is a random isogeny of degree $q(2^{a-2} - q)$ for some random q, and

$$(P_A, Q_A, R_A, S_A, X_A) = (\varphi(P_0), \varphi(Q_0), \varphi(R_0), \varphi(S_0), \varphi(X_0))$$

repeat
  $q \xleftarrow{\$} \{0, \ldots, 2^{a-2} - 1\}$
until $\gcd(q(2^{a-2} - q),\ 2 \cdot C \cdot D) = 1$
$\theta \leftarrow$ GeneralizedRepresentInteger$(q(2^{a-2} - q)C)$
$(P_0^\tau, Q_0^\tau) \leftarrow (\theta(P_0), \theta(Q_0))$
$K \leftarrow$ EndomorphismToKernel$(\overline{\theta})$
$\widetilde{E_A}, (\widetilde{P_A}, \widetilde{Q_A}) \leftarrow$ OddIsogenyChain$(E_0, K, C, (P_0^\theta, Q_0^\theta))$
$s \leftarrow Cq \bmod 2^a$
$\_, ((\widetilde{P_A}, \widetilde{P'_A}), (\widetilde{Q_A}, \widetilde{Q'_A}), (\widetilde{R_A}, \_), (\widetilde{S_A}, \_), (\widetilde{X_A}, \_)) \leftarrow$
  Isogeny22Chain$(([s]P_0, \widetilde{P_A}), ([s]Q_0, \widetilde{Q_A}), [(P_0, 0), (Q_0, 0), (R_0, 0), (S_0, 0), (X_0, 0)])$
$(E_A, \_), ((P_A, \_), (Q_A, \_), (R_A, \_), (S_A, \_), (X_A, \_)) \leftarrow$
  Isogeny22Chain$((\widetilde{P_A}, \widetilde{P'_A}), (\widetilde{Q_A}, \widetilde{Q'_A}), [(\widetilde{P_A}, 0), (\widetilde{Q_A}, 0), (\widetilde{R_A}, 0), (\widetilde{S_A}, 0), (\widetilde{X_A}, 0)])$
return $(q, E_A, (P_A, Q_A), (R_A, S_A), X_A)$

- the retained secret key is $\mathsf{sk} = (q, \alpha_2, \beta_2, \delta_D)$;
- the public key is $\mathsf{pk} = (E_A, P_A, Q_A, R_A, S_A, X_A)$.

The scalar γ_C (γ_1, γ_2 in the following pseudocode) is sampled during key generation and is used only to mask the published C-torsion images. It is not required to be stored as a part of the long-term secret key.

3.2.2 Encryption

The encryption procedure is specified in QIMEN-PIKE.PKE.Encrypt. It computes isogenies

$$\psi : E_0 \to E_B \qquad \text{and} \qquad \psi' : E_A \to E_{AB},$$

such that

$$\ker(\psi) = \langle R_0 + [r_3]S_0 \rangle \qquad \text{and} \qquad \ker(\psi') = \langle R_A + [r_3]S_A \rangle,$$

for some $r_3 \in \mathbb{Z}/C\mathbb{Z}$.

The procedure then computes the masked evaluations of these isogenies on the published torsion data, namely

$$P_B, Q_B, Y_B \in E_B(\overline{\mathbb{F}}_p) \qquad \text{and} \qquad P_{AB}, Q_{AB}, X_{AB} \in E_{AB}(\overline{\mathbb{F}}_p),$$

Algorithm 8 EndomorphismToKernel(α)

Input: An endomorphism $\alpha \in \mathcal{O}_0 \simeq \mathrm{End}(E_0)$ with $C \mid \deg(\alpha)$
Output: A kernel generator K of the subgroup $\{K \in E_0[C] \mid \alpha(K) = 0_{E_0}\}$

Let $\mathbf{A} = [1, \mathbf{i}, \frac{\mathbf{i}+\mathbf{j}}{2}, \frac{1+\mathbf{k}}{2}]$ denote the fixed integral basis of $\mathcal{O}_0$
Compute $\mathbf{v}_\alpha = [x_1, x_2, x_3, x_4]^T \in \mathbb{Z}^4$ such that $\mathbf{A}\mathbf{v}_\alpha = \alpha$
$\mathbf{M}_\alpha = x_1 M_1 + x_2 M_2 + x_3 M_3 + x_4 M_4$
Compute $[v_1, v_2]^T \in \ker(\mathbf{M}_\alpha)$
$K \leftarrow [v_1]R_0 + [v_2]S_0$
return K

Algorithm 9 QIMEN-PIKE.PKE.KeyGen(λ)

Input: The internal security parameter λ
Output: The secret key and the public key $(\mathsf{sk}, \mathsf{pk})$

$(q, E_A, (P'_A, Q'_A), (R'_A, S'_A), X'_A) \leftarrow$ QIMEN-PIKE.CoreKeyGenIso(pp)
$\alpha_2, \beta_2 \xleftarrow{\$} (\mathbb{Z}/2^{a-2}\mathbb{Z})^\times$, $\gamma \xleftarrow{\$} (\mathbb{Z}/C\mathbb{Z})^\times$, $\delta_D \xleftarrow{\$} (\mathbb{Z}/D\mathbb{Z})^\times$
$P_A, Q_A \leftarrow [\alpha_2]P'_A, [\beta_2]Q'_A$
$R_A, S_A \leftarrow [\gamma]R'_A, [\gamma]S'_A$
$X_A \leftarrow [\delta_D]X'_A$
$\mathsf{sk} \leftarrow (q, \alpha_2, \beta_2, \delta_D)$
$\mathsf{pk} \leftarrow (E_A, (P_A, Q_A), (R_A, S_A), X_A)$
return $(\mathsf{sk}, \mathsf{pk})$

as specified in Figure 3.1.

The ciphertext consists of the codomain curves E_B and E_{AB}, the six masked image points above, and the masked message $\mathsf{pad} \oplus \mathsf{msg}$. The pad pad is derived from the shared D-torsion value determined by the masking scalars η_D and ζ_D.

In Step 3 above, the pad is derived from the real part of $w_0^{\eta_D \zeta_D}$ rather than from $w_0^{\eta_D \zeta_D}$ itself. This choice is tied to the ciphertext encoding. When only the x-coordinates of the transmitted torsion points are retained, the corresponding pairing value recovered during decryption is determined only up to inversion. Taking the trace removes the ambiguity, since $\mathrm{tr}(w_0^{\eta_D \zeta_D}) = \mathrm{tr}(w_0^{-\eta_D \zeta_D}) = w_0^{\eta_D \zeta_D} + w_0^{-\eta_D \zeta_D}$. As a result, the value used in the key-derivation step lies in $\mathbb{F}_p$ rather than in $\mathbb{F}_{p^2}$.

3.2.3 Decryption

The decryption procedure derives the same pad pad as in Step 3 of QIMEN-PIKE.PKE.Encrypt. To do so, the receiver uses the ciphertext points $Y_B \in E_B[D]$ and $X_{AB} \in E_{AB}[D]$, together with the secret key $(q, \alpha_2, \beta_2, \delta_D)$, to recover the corresponding pairing value.

More precisely, decryption first reconstructs a 2-dimensional isogeny from the 2^a-torsion data $(P_B, Q_B) \in E_B[2^a]$ and $(P_{AB}, Q_{AB}) \in E_{AB}[2^a]$. Applying this isogeny to Y_B and X_{AB}

Algorithm 10 QIMEN-PIKE.PKE.ENCRYPT($\mathsf{pk}, \mathsf{msg}$)

Input: The public key $\mathsf{pk} = (E_A, P_A, Q_A, R_A, S_A, X_A)$ and a message $\mathsf{msg} \in \{0,1\}^{2\lambda}$
Output: The ciphertext ct

$r \xleftarrow{\$} \mathbb{Z}/C\mathbb{Z}$, $\omega_2 \xleftarrow{\$} (\mathbb{Z}/2^a\mathbb{Z})^\times$, $\eta_D, \zeta_D \xleftarrow{\$} (\mathbb{Z}/D\mathbb{Z})^\times$
$w' \leftarrow w_0^{\eta_D \zeta_D}$
$\mathsf{pad} \leftarrow \mathsf{H}(\mathsf{KDF} \| \mathrm{tr}(w'))$
$E_B, (P'_B, Q'_B, Y'_B) \leftarrow \textsc{OddIsogenyChain}(E_0, R_0 + [r]S_0, C, (P_0, Q_0, Y_0))$
$E_{AB}, (P'_{AB}, Q'_{AB}, X'_{AB}) \leftarrow \textsc{OddIsogenyChain}(E_A, R_A + [r]S_A, C, (P_A, Q_A, X_A))$
$P_B, Q_B, Y_B \leftarrow [\omega_2]P'_B,\ [\omega_2^{-1} \bmod 2^a]Q'_B,\ [\eta_D]Y'_B$
$P_{AB}, Q_{AB}, X_{AB} \leftarrow [\omega_2]P'_{AB},\ [\omega_2^{-1} \bmod 2^a]Q'_{AB},\ [\zeta_D]X'_{AB}$
$c \leftarrow \mathsf{pad} \oplus \mathsf{msg}$
$\mathsf{ct} \leftarrow (E_B,\ P_B,\ Q_B,\ Y_B,\ E_{AB},\ P_{AB},\ Q_{AB},\ X_{AB},\ c)$
return ct

yields two points on a common intermediate curve, denoted by Y_M and X_M in QIMEN-PIKE.PKE.DECRYPT. Their pairing gives a known power of the shared value used in encryption. The exponent differs from the desired one by the factor $q(2^{a-2} - q)C\delta_D$, which is known to the receiver. Therefore, the same shared value is recovered by raising the pairing output to the inverse of this factor modulo D.

In QIMEN-PIKE, the shared-value computation is carried out through the intermediate curve recovered during decryption. This avoids reconstructing $\phi(Y_B)$ explicitly on E_{AB} and matches the key-generation and encryption formats defined above. The complete decryption procedure is specified in QIMEN-PIKE.PKE.DECRYPT.

Algorithm 11 QIMEN-PIKE.PKE.DECRYPT(sk, ct)

Input: The secret key $\mathsf{sk} = (q, \alpha_2, \beta_2, \delta_D)$ and the ciphertext $\mathsf{ct} = (E_B, P_B, Q_B, Y_B, E_{AB}, P_{AB}, Q_{AB}, X_{AB}, c)$
Output: The message msg

$\widetilde{P_B} \leftarrow [-q]P_B, \widetilde{Q_B} \leftarrow [-q]Q_B$
$\widetilde{P_{AB}} \leftarrow [\alpha_2^{-1} \bmod 2^a]P_{AB}, \widetilde{Q_{AB}} \leftarrow [\beta_2^{-1} \bmod 2^a]Q_{AB}$
$E_M, \_, (Y_M, \_), (X_M, \_)$
$\quad \leftarrow \textsc{Isogeny22Chain}((\widetilde{P_B}, \widetilde{P_{AB}}), (\widetilde{Q_B}, \widetilde{Q_{AB}}), [(Y_B, 0_{E_B}), (0_{E_{AB}}, X_{AB})])$
$t \leftarrow \left(q(2^{a-2} - q)C\delta_D\right)^{-1} \bmod D$
$w \leftarrow \textsc{TateOdd}(E_M, D, X_M, Y_M, X_M - Y_M)$
$\widetilde{w'} \leftarrow w^t$
$\widetilde{\mathsf{pad}} \leftarrow \mathsf{H}(\mathsf{KDF} \| \mathrm{tr}(w'))$
return $\mathsf{pad} \oplus c$

Algorithm 12 QIMEN-PIKE.Enc.Det($\mathsf{pk}, \mathsf{msg}, \rho$)

Input: The public key pk, a message $\mathsf{msg} \in \{0,1\}^{2\lambda}$, and a seed $\rho \in \{0,1\}^{\mu}$.
Output: The ciphertext ct

1: $(r_3, \omega_2, \eta_D, \zeta_D) \leftarrow \mathsf{DeriveCoins}(\rho)$
2: Run QIMEN-PIKE.PKE.Encrypt with these values replacing the random sampling in Step 1
3: **return** the resulting ciphertext ct

3.3 Specification of PIKE.KEM

From the IND-CPA secure public-key encryption scheme QIMEN-PIKE.PKE, we construct the IND-CCA2 secure key encapsulation mechanism QIMEN-PIKE.KEM via a Fujisaki–Okamoto-style transform. We present QIMEN-PIKE.KEM as a combination of three algorithms, QIMEN-PIKE.KeyGen, QIMEN-PIKE.Encaps, and QIMEN-PIKE.Decaps.

The KEM secret key stores the underlying PKE secret key together with the public key, a hash of the public key, and a fallback secret used for implicit rejection.

3.3.1 Auxiliary functions for the KEM layer

For the KEM layer, we use the following domain-separated functions.

$\mathsf{H}_{\mathsf{pk}} : \{0,1\}^{\star} \to \{0,1\}^{2\lambda}$ A hash of the public key. It is used in encapsulation and stored as part of the KEM secret key.

$\mathsf{H}_{\mathsf{ct}} : \{0,1\}^{\star} \to \{0,1\}^{2\lambda}$ A hash of the ciphertext. It binds the final shared secret to the full ciphertext transcript.

$\mathsf{G} : \{0,1\}^{\star} \to \{0,1\}^{2\lambda} \times \{0,1\}^{\mu}$ A hash-and-expand function. On input $m \| h_{\mathsf{pk}}$, it outputs a pre-key $\bar{K} \in \{0,1\}^{2\lambda}$ and a seed $\rho \in \{0,1\}^{\mu}$ for deterministic encapsulation.

$\mathsf{KDF} : \{0,1\}^{\star} \to \{0,1\}^{N_{\mathsf{ss}}}$ A key-derivation function used to derive the final shared secret.

$\mathsf{DeriveCoins} : \{0,1\}^{\mu} \to \mathbb{Z}/C\mathbb{Z} \times (\mathbb{Z}/2^a\mathbb{Z})^{\times} \times (\mathbb{Z}/D\mathbb{Z})^{\times} \times (\mathbb{Z}/D\mathbb{Z})^{\times}$ A deterministic parser that expands ρ and derives the tuple $(r_3, \omega_2, \eta_D, \zeta_D)$ used by encapsulation.

The function $\mathsf{DeriveCoins}$ is defined by expanding ρ into a byte stream and parsing that stream by rejection sampling until values in the required domains are obtained, namely $r_3 \in \mathbb{Z}/C\mathbb{Z}$, $\omega_2 \in (\mathbb{Z}/2^a\mathbb{Z})^{\times}$, $\eta_D \in (\mathbb{Z}/D\mathbb{Z})^{\times}$, and $\zeta_D \in (\mathbb{Z}/D\mathbb{Z})^{\times}$.

3.3.2 Algorithms

The full description of QIMEN-PIKE.KEM is specified in the following algorithms.

Algorithm 13 QIMEN-PIKE.KeyGen(λ)

Input: The internal security parameter λ.
Output: The public key pk and the secret key sk.

$z \leftarrow \{0,1\}^{2\lambda}$.
$(\mathsf{pk}, \mathsf{sk}_0) \leftarrow$ QIMEN-PIKE.PKE.KeyGen(λ).
$h_{\mathsf{pk}} \leftarrow \mathsf{H}_{\mathsf{pk}}(\mathsf{pk})$.
$\mathsf{sk} \leftarrow (\mathsf{sk}_0 \| \mathsf{pk} \| h_{\mathsf{pk}} \| z)$.
return $(\mathsf{pk}, \mathsf{sk})$.

Algorithm 14 QIMEN-PIKE.Encaps(pk)

Input: The public key pk.
Output: A ciphertext ct and a shared secret K.

$m \leftarrow \{0,1\}^{2\lambda}$.
$h_{\mathsf{pk}} \leftarrow \mathsf{H}_{\mathsf{pk}}(\mathsf{pk})$.
$(\bar{K}, \rho) \leftarrow \mathsf{G}(m \| h_{\mathsf{pk}})$.
$\mathsf{ct} \leftarrow$ QIMEN-PIKE.Enc.Det(pk, m, ρ).
$K \leftarrow \mathsf{KDF}(\bar{K} \| \mathsf{H}_{\mathsf{ct}}(\mathsf{ct}))$.
return (ct, K).

Algorithm 15 QIMEN-PIKE.Decaps(ct, sk)

Input: The ciphertext ct and the secret key $\mathsf{sk} = (\mathsf{sk}_0 \| \mathsf{pk} \| h_{\mathsf{pk}} \| z)$.
Output: A shared secret K.

try
    $m' \leftarrow$ QIMEN-PIKE.PKE.Decrypt($\mathsf{ct}, \mathsf{sk}_0$).
catch
    return $K \leftarrow \mathsf{KDF}(z \| \mathsf{H}_{\mathsf{ct}}(\mathsf{ct}))$.
$(\bar{K}', \rho') \leftarrow \mathsf{G}(m' \| h_{\mathsf{pk}})$.
$\mathsf{ct}' \leftarrow$ QIMEN-PIKE.Enc.Det(pk, m', ρ').
if $\mathsf{ct} = \mathsf{ct}'$ **then**
    return $K \leftarrow \mathsf{KDF}(\bar{K}' \| \mathsf{H}_{\mathsf{ct}}(\mathsf{ct}))$.
else
    return $K \leftarrow \mathsf{KDF}(z \| \mathsf{H}_{\mathsf{ct}}(\mathsf{ct}))$.

CHAPTER 4

Size compression

This section introduces our techniques for optimizing the sizes of the public key and ciphertext in QIMEN-PIKE. We propose two approaches: the first approach reduces the number of stored points via point combination, while the second approach further compresses the point information by storing their scalar representations with respect to fixed torsion bases. Compared to the first approach, the second trades computational efficiency for a more compact representation. Correspondingly, we derive two implementations: the uncompressed implementation and the compressed implementation.

4.1 Uncompressed implementation

Recall from Chapter 3 that the public key is of the form $\mathsf{pk} = \{E_A, (P_A, Q_A), (R_A, S_A), X_A\}$, where (P_A, Q_A) is a 2^a-torsion basis, (R_A, S_A) is a C-torsion basis, and X_A is a point of order D. All rational points are defined over E_A. In the following, we first discuss how to reduce the public key size by combining these points.

Note that $C = C_1C_2$, where $C_1 \mid p+1$ and $C_2 \mid p-1$. Given a point R of order C, we define $R^1 = [C_2]R$ and $R^2 = [C_1]R$. For efficiency, during the setup phase we define $E_0[C_1] = \langle R_0^1, S_0^1 \rangle \subset E_0(\mathbb{F}_{p^2})$ and $E_0[C_2] = \langle R_0^2, S_0^2 \rangle \subset E_0(\mathbb{F}_{p^4})$ instead of $E_0[C] = \langle R_0, S_0 \rangle \subset E_0(\mathbb{F}_{p^4})$. Although $E_0[C_2] = \langle P_0^2, Q_0^2 \rangle \subset E_0(\mathbb{F}_{p^4})$, the x-coordinates of the points in $E_0[C_2]$ are defined over $\mathbb{F}_{p^2}$, enabling computations to be performed over $\mathbb{F}_{p^2}$ using x-only arithmetic.

For this reason, we add to the public parameters the following values:

- P_0^+: the point P_0^+ represented by its x-coordinate, satisfying $[C_1D]P_0^+ = P_0$, $[2^aD]P_0^+ = R_0^1$ and $[2^aC_1]P_0^+ = X_0$;
- Q_0^+: the point Q_0^+ represented by its x-coordinate, satisfying $[C_1]Q_0^+ = Q_A$ and $[2^a]Q_0^+ = S_A^1$;
- R_0^-: the point $R_0^- = R_0^2$ represented by its x-coordinate;
- S_0^-: the point $S_0^- = S_0^2$ represented by its x-coordinate;

- T_0^+: the point $T_0^+ = [D]P_0^+ - Q_0^+$ represented by its x-coordinate
- T_0^-: the point $T_0^- = R_0^- - S_0^-$ represented by its x-coordinate

4.1.1 Public key

To optimize the public key size, the uncompressed implementation formats the public key as

$$(P_A^+, Q_A^+, R_A^-, S_A^-, T_A^-, \mathsf{label}_{\mathsf{pk}}).$$

Precisely,

- P_A^+: the point P_A^+ represented by its x-coordinate, satisfying $[C_1 D]P_A^+ = P_A$, $[2^a D]P_A^+ = R_A^1$ and $[2^a C_1]P_A^+ = X_A$;
- Q_A^+: the point Q_A^+ represented by its x-coordinate, satisfying $[C_1]Q_A^+ = Q_A$ and $[2^a]Q_A^+ = S_A^1$;
- R_A^-: the point $R_A^- = R_A^2$ represented by its x-coordinate;
- S_A^-: the point $S_A^- = S_A^2$ represented by its x-coordinate;
- T_A^-: the point $T_A^- = R_A^2 - S_A^2$ represented by its x-coordinate;
- $\mathsf{label}_{\mathsf{pk}}$: a boolean flag used to recover the x-coordinate of the point $[D]P_A^+ - Q_A^+$, satisfying that $[2^a]([D]P_A^+ - Q_A^+) = R_A^1 - S_A^1$ and $[C_1]([D]P_A^+ - Q_A^+) = P_A - Q_A$.

Since P_A, Q_A, R_A^1, S_A^1, and X_A are defined over $E_A(\mathbb{F}_{p^2})$, the points P_A^+ and Q_A^+ are also defined over $E_A(\mathbb{F}_{p^2})$. In total, the public key comprises five $\mathbb{F}_{p^2}$ elements and a single boolean bit.

To summarize, Algorithm 17 illustrates the key generation procedures of our uncompressed implementation. The sub-algorithm QIMEN-PIKE.CoreKeyGenIso is also modified correspondingly, as proposed in Algorithm 16.

Remark 4.1.1. *Note that at Steps 11 and 14 of QIMEN-PIKE.ModifiedCoreKeyGenIso, the 2-dimensional isogenies evaluates not only the four points P_0^+, Q_0^+, R_0^- and S_0^-, but also $T_0^+ = [D]P_A^+ - Q_0^+$ and T_0^-. One might ask why we additionally evaluate the difference points, given that T_A^+ (resp. T_A^-) can be derived from P_{AB}^+, Q_{AB}^+ (resp. R_A^-, S_A^-). The reason lies in the implementation's use of x-only arithmetic. Although computing the point addition $[D]P_A^+ - Q_0^+$ is feasible, distinguishing between $[D]P_A^+ - Q_0^+$ and $[D]P_A^+ + Q_0^+$ is inefficient when only the x-coordinates are available. This same rationale explains why the algorithm evaluates the 2-dimensional isogenies at T_0^-.*

4.1.2 Ciphertext

Next, we consider how to combine the point information within the ciphertext. The ciphertext involves two elliptic curves E_B and E_{AB}, the 2^a-torsion bases $(P_B, Q_B) \in E_B[2^a]$ and $(P_{AB}, Q_{AB}) \in E_{AB}[2^a]$, and two points $Y_B \in E_B[D]$ and $X_{AB} \in E_{AB}[D]$. In the

Algorithm 16 QIMEN-PIKE.ModifiedCoreKeyGenIso(λ)

Input: The internal security parameter λ

Output: $(q, E_A, P_A^+, Q_A^+, T_A^+, R_A^-, S_A^-, T_A^-)$ such that $q(2^{a-2}-q)$ is the degree of a random isogeny $\varphi : E_0 \to E_A$, and

$$(P_A^+, Q_A^+, T_A^+, R_A^-, S_A^-, T_A^-) = (\varphi(P_0^+), \varphi(Q_0^+), \varphi(T_0^+), \varphi(R_0^-), \varphi(S_0^-), \varphi(T_0^-))$$

repeat
  $q \xleftarrow{\$} \{0, \ldots, 2^{a-2}-1\}$
until $\gcd(q(2^{a-2}-q),\ 2 \cdot C \cdot D) = 1$
$\theta \leftarrow$ GeneralizedRepresentInteger$(q(2^{a-2}-q)C)$
$(P_0^\theta, Q_0^\theta) \leftarrow (\theta(P_0), \theta(Q_0))$
$K_1 \leftarrow$ EndomorphismToKernel$(E_0, (R_0^1, S_0^1), C_1, \overline{\theta})$
$K_2 \leftarrow$ EndomorphismToKernel$(E_0, (R_0^2, S_0^2), C_2, \overline{\theta})$
$\widetilde{E_A}, (\widetilde{P_A}, \widetilde{Q_A}, \widetilde{K_2}) \leftarrow$ OddIsogenyChain$(E_0, K_1, C_1, (P_0^\theta, Q_0^\theta, K_2))$
$\widetilde{E_A}, (\widetilde{P_A}, \widetilde{Q_A}) \leftarrow$ OddIsogenyChain$(\widetilde{E_A}, \widetilde{K_2}, C_2, (\widetilde{P_A}, \widetilde{Q_A}))$
$\texttt{pts} \leftarrow [(P_0, 0), (Q_0, 0), (P_0^+, 0), (Q_0^+, 0), (T_0^+, 0), (R_0^-, 0), (S_0^-, 0), (T_0^-, 0)]$
$\_, \texttt{pts} \leftarrow$ Isogeny22Chain$(([Cq]P_0, \widetilde{P_A}), ([Cq]Q_0, \widetilde{Q_A}), \texttt{pts})$
Parse pts as $((\widetilde{P_A}, \widetilde{P'_A}), (\widetilde{Q_A}, \widetilde{Q'_A}), (\widetilde{P_A^+}, \_), (\widetilde{Q_A^+}, \_), (\widetilde{T_A^+}, \_), (\widetilde{R_A^-}, \_), (\widetilde{S_A^-}, \_), (\widetilde{T_A^-}, \_))$
$\texttt{pts} \leftarrow [(\widetilde{P_A^+}, 0), (\widetilde{Q_A^+}, 0), (\widetilde{T_A^+}, 0)), (\widetilde{R_A^-}, 0), (\widetilde{S_A^-}, 0), (\widetilde{T_A^-}, 0)]$
$(E_A, \_), ((P_A^+, \_), (Q_A^+, \_), (T_A^+, \_), (R_A^-, \_), (S_A^-, \_), (T_A^-, \_)) \leftarrow$
  Isogeny22Chain$((\widetilde{P_A}, \widetilde{P'_A}), (\widetilde{Q_A}, \widetilde{Q'_A}), \texttt{pts})$
return $(q, E_A, P_A^+, Q_A^+, T_A^+, R_A^-, S_A^-, T_A^-)$

uncompressed implementation, the ciphertext takes the form

$$(E_B, E_{AB}, P_B^+, Q_B^+, P_{AB}^+, Q_{AB}^+, c), \qquad c = \mathsf{pad} \oplus \mathsf{msg},$$

where

- E_B: the elliptic curve E_B represented by its Montgomery coefficient;
- E_{AB}: the elliptic curve E_{AB} represented by its Montgomery coefficient;
- P_B^+: the point $P_B^+ = P_B$ represented by its x-coordinate;
- Q_B^+: the point Q_B^+ represented by its x-coordinate, satisfying $[D]Q_B^+ = Q_B$ and $[2^a]Q_B^+ = Y_B$;
- P_{AB}^+: the point P_{AB}^+ represented by its x-coordinate, satisfying $[D]P_{AB}^+ = P_{AB}$ and $[2^a]P_{AB}^+ = X_{AB}$;
- Q_{AB}^+: the point $Q_{AB}^+ = Q_{AB}$ represented by its x-coordinate.

Since all the considered points are defined over $E_B(\mathbb{F}_{p^2})$ or $E_{AB}(\mathbb{F}_{p^2})$, their x-coordinates are defined over $\mathbb{F}_{p^2}$. Therefore, the ciphertext involves six $\mathbb{F}_{p^2}$-elements.

Algorithm 17 QIMEN-PIKE.KeyGenUncompressed(λ)

Input: The internal security parameter λ
Output: The secret key and the public key $(\mathsf{sk}, \mathsf{pk})$

1: $(q, E_A, P_A^+, Q_A^+, T_A^+, R_A^-, S_A^-, T_A^-) \leftarrow$ QIMEN-PIKE.ModifiedCoreKeyGenIso()
2: $\alpha_2, \beta_2 \xleftarrow{\$} (\mathbb{Z}/2^{a-2}\mathbb{Z})^\times$, $\gamma_1 \xleftarrow{\$} (\mathbb{Z}/C_1\mathbb{Z})^\times$, $\gamma_2 \xleftarrow{\$} (\mathbb{Z}/C_2\mathbb{Z})^\times$, and $\delta_D \xleftarrow{\$} (\mathbb{Z}/D\mathbb{Z})^\times$
3: Compute t_1 such that $t_1 \bmod 2^a \equiv \alpha_2$, $t_1 \bmod C_1 \equiv \gamma_1$ and $t_1 \bmod D \equiv \delta_D$
4: Compute t_2 such that $t_2 \bmod 2^a \equiv \beta_2$ and $t_2 \bmod C_1 \equiv \gamma_1$
5: $T_3 \leftarrow$ LadderBiscalar$([D]P_A^+, Q_A^+, T_A^+, E_A, t_1, t_2)$
6: $P_A^+ \leftarrow [t_1]P_A^+, Q_A^+ \leftarrow [t_2]Q_A^+$
7: $x_{P_A^+} \leftarrow x(P_A^+)/z(P_A^+)$, $x_{Q_A^+} \leftarrow x(Q_A^+)/z(Q_A^+)$
8: $(x_{[D]P_A^+} : z_{[D]P_A^+}) \leftarrow$ Ladder$((x_{P_A^+} : 1), E_A, D)$
9: $T_1 \leftarrow$ ProjectiveDifference$((x_{[D]P_A^+} : z_{[D]P_A^+}), (x_{Q_A^+} : 1), E_A)$
10: $\mathsf{label}_{\mathsf{pk}} \leftarrow (T_1 = T_3)$
11: $R_A^- \leftarrow [\gamma_2]R_A^-$, $S_A^- \leftarrow [\gamma_2]S_A^-$, $T_A^- \leftarrow [\gamma_2]T_A^-$
12: $\mathsf{sk} \leftarrow (q, \alpha_2, \beta_2, \delta_D)$
13: $\mathsf{pk} \leftarrow (x_{P_A^+}, x_{Q_A^+}, x(R_A^-)/z(R_A^-), x(S_A^-)/z(S_A^-), x(T_A^-)/z(T_A^-), \mathsf{label}_{\mathsf{pk}})$
14: **return** $(\mathsf{sk}, \mathsf{pk})$

Algorithms 18 and 19 summarize the encryption and decryption procedures, respectively.

Remark 4.1.2. *In the uncompressed implementation, the public key does not explicitly include the elliptic curve E_A. However, the curve coefficients remain necessary during encryption for scalar multiplication and ladder algorithms. For instance, the sender must execute* Ladder3pt *to compute the kernel of the isogeny $\psi' : E_A \to E_{AB}$, or use* ProjectiveDifference *to determine the x-coordinate of $[D]P_A^+ - Q_A^+$. To facilitate this, the elliptic curve can be uniquely recovered using* RecoverCodomain *at Step 1 of Algorithm 18, taking the points R_A^-, S_A^-, and their difference T_A^- as input.*

4.2 Compressed implementation

With the exception of the D-torsion points, all other points possess a smooth order, facilitating further compression. For a given point $P \in E$ of smooth order N, one can generate a deterministic basis (U, V) of the same order and express P as a linear combination of this basis:

$$P = [s]U + [t]V. \tag{4.1}$$

By storing the scalars (s, t) rather than the coordinates of P, the storage requirement is significantly reduced from $2\log p$ bits to $2\log N$ bits, where $s, t \in \mathbb{Z}/N\mathbb{Z}$. Furthermore,

because P, U, and V all have full order N, at least one of the scalars s or t must be invertible modulo N. In scenarios where it suffices to transmit a generator of the subgroup $\langle P \rangle$ instead of P itself, one can transmit $[s^{-1}]P$ (or $[t^{-1}]P$ if s is not invertible). Consequently, the scalar pair (s, t) can be normalized and further compressed to $(1, s^{-1}t)$ or $(s^{-1}t, 1)$.

Following Eq. (4.1), recovering the scalars s and t from P necessitates solving a two-dimensional elliptic curve discrete logarithm. While the smoothness of N ensures this can be solved in polynomial time, the process remains computationally demanding due to the extensive scalar multiplications and point additions required. To mitigate this overhead, one can employ cryptographic pairings. Observing that

$$\begin{aligned} h_0 &= t_N(U, V), \\ h_1 &= t_N(U, P) = t_N(U, [s]U + [t]V) = t_N(U, V)^t, \\ h_2 &= t_N(V, P) = t_N(V, [s]U + [t]V) = t_N(U, V)^{-s}, \end{aligned} \tag{4.2}$$

pairing evaluations allow us to map the problem into the multiplicative group $\mu_N = \{x \in \mathbb{F}_{p^2} \mid x^N = 1\}$. By doing so, the scalars s and t can be efficiently extracted via discrete logarithms in μ_N, circumventing the need to directly compute the two-dimensional elliptic curve discrete logarithm.

Applying a similar methodology to an entire N-torsion basis (P, Q), one can compress the basis into $(s, t, u, v) \in (\mathbb{Z}/N\mathbb{Z})^4$. Because (P, Q) forms a valid basis, the corresponding transformation matrix is invertible, guaranteeing that a valid scalar inversion exists. Consequently, this representation can be normalized and further compressed to exactly three scalars.

4.2.1 Public key

Recall that the public key consists of an elliptic curve E_A, a 2^a-torsion basis (P_A, Q_A), a C_1-torsion basis (R_A^1, S_A^1), a C_2-torsion basis (R_A^2, S_A^2), and a point X_A of order D. Because D is non-smooth, X_A cannot be efficiently compressed via discrete logarithm computations. Consequently, the public key must include at least one point in x-only coordinates to represent X_A. Conversely, owing to the smoothness of 2^a, C_1, and C_2, we can express these bases as follows:

$$\begin{aligned} \begin{pmatrix} P_A \\ Q_A \end{pmatrix} &= \begin{pmatrix} s_{A,2^a} & t_{A,2^a} \\ u_{A,2^a} & v_{A,2^a} \end{pmatrix} \cdot \begin{pmatrix} U_{A,2^a} \\ V_{A,2^a} \end{pmatrix}, \\ \begin{pmatrix} R_A^1 \\ S_A^1 \end{pmatrix} &= \begin{pmatrix} s_{A,C_1} & t_{A,C_1} \\ u_{A,C_1} & v_{A,C_1} \end{pmatrix} \cdot \begin{pmatrix} U_{A,C_1} \\ V_{A,C_1} \end{pmatrix}, \\ \begin{pmatrix} R_A^2 \\ S_A^2 \end{pmatrix} &= \begin{pmatrix} s_{A,C_2} & t_{A,C_2} \\ u_{A,C_2} & v_{A,C_2} \end{pmatrix} \cdot \begin{pmatrix} U_{A,C_2} \\ V_{A,C_2} \end{pmatrix}, \end{aligned}$$

where $(U_{A,2^a}, V_{A,2^a})$, (U_{A,C_1}, V_{A,C_1}) and (U_{A,C_2}, V_{A,C_2}) are deterministic 2^a-, C_1- and C_2-torsion bases on E_A, respectively.

Fortunately, the scalar representations of all torsion bases can be compressed into three scalars each without affecting the encryption procedure. Naively, this representation requires $2\log p$ bits to store the curve E_A, $2\log p$ bits to store the point X_A, and $3a$, $3\log C_1$, and $3\log C_2$ bits to store (P_A, Q_A), (R_A^1, S_A^1), and (R_A^2, S_A^2), respectively. Let $C_1 = \prod_{i=1}^{n} \ell_i^{n_i}$. To fully exploit the capacity of x-only projective coordinates, we instead store a single composite point P_{AB}^+ of order $2^a C_1 D$, such that

$$[C_1 D]P_A^+ = [m_{A,2^a}]P_A, [2^a D]P_A^+ = [m_{A,C_1}]R_A^1 \text{ and } [2^a C_1]P_A^+ = X_A,$$

where

$$m_{A,2^a} = \begin{cases} v_{A,2^a}^{-1} \bmod 2^a, & \text{if } v_{A,2^a} \text{ is invertible in } \mathbb{Z}/2^a\mathbb{Z}\ , \\ u_{A,2^a}^{-1} \bmod 2^a, & \text{otherwise,} \end{cases}$$

and

$$m_{A,C_1} = \begin{cases} u_{A,C_1}^{-1} \bmod \ell_i^{n_i}, & \text{if } u_{A,C_1} \text{ is invertible in } \mathbb{Z}/\ell_i^{n_i}\mathbb{Z}\ , \\ v_{A,C_1}^{-1} \bmod \ell_i^{n_i}, & \text{otherwise,} \end{cases}$$

with $i = 1, 2, \ldots, n$.

By including the x-coordinate of this composite point P_A^+ in the public key, the necessity to explicitly store the scalar representations associated with P_A and R_A^1 is entirely eliminated. Upon receiving $x(P_A^+)$, the sender can independently recover the x-coordinates of $[C_1 D]P_A^+$ and $[2^a D]P_A^+$. Consequently, the information of P_A and R_A^1 is implicitly transmitted without requiring any extra bits. This optimization amortizes the heavy cost of storing X_A in uncompressed x-only coordinates, allowing the remaining basis points (such as Q_A and S_A^1) to be represented with a minimal number of scalars, thereby yielding a highly compact public key.

We also adapt some tricks to minimize the comrpessed public key size. In our setup, we choose Q_0 as a 2^a-torsion point satisfying $[2^{a-1}]Q_0 = (0, 0)$, which is a point of order 2 on Montgomery curves. Since both isogeny evaluation algorithms (OddIsogenyChain and Isogeny22Chain) map $(0, 0)$ on the domain curve to $(0, 0)$ on the codomain curve, and the isogeny degrees are odd (meaning they act as isomorphisms on the 2-torsion groups), all considered isogenies map P_0 to a point away from $(0, 0)$ and Q_0 to a point over $(0, 0)$. By ensuring that the deterministic 2^a-torsion basis $\{U_{A,2^a}, V_{A,2^a}\}$ on E_A shares the same structural property, i.e., $[2^{a-1}]V_{A,2^a} = (0, 0)$, the coefficient $v_{A,2^a}$ is guaranteed to be invertible in $\mathbb{Z}/2^a\mathbb{Z}$. This allows us to definitively set $m_{A,2^a} = (v_{A,2^a})^{-1} \bmod 2^a$, thereby eliminating the need for a boolean flag in the public key to distinguish between $m_{A,2^a} = (v_{A,2^a})^{-1} \bmod 2^a$ and $m_{A,2^a} = (u_{A,2^a})^{-1} \bmod 2^a$. This technique applies equally well to ciphertext compression. Furthermore, we restrict C_2 to be a prime power, ensuring that the compressed C_2-torsion information does not require an additional identifier such as label_{C_1}.

To summarize, the compressed public key is of the form

$$\mathsf{pk} = (E_A, P_A^+, \mathsf{hint}_{C_1}, \mathsf{hint}_{C_2}, s_{A,2^a}, s_{C_1}, \mathsf{label}_{C_1}, \mathsf{label}_+, s_{C_2}^1, s_{C_2}^2, s_{C_2}^3, \mathsf{label}_{C_2}),$$

where:

- E_A: the elliptic curve E_A represented by its Montgomery coefficient;
- P_A^+: the point P_A^+ represented by its x-coordinate, satisfying $[D]P_A^+ = [m_{2^a}]P_A$, $[2^a D]P_A^+ = [m_{C_1}]R_A^1$ and $[2^a C_1]P_A^+ = X_A$;
- $\mathsf{hint}_{2^a C_1}$: a hint used to compute the deterministic $2^a C_1$-torsion basis on E_A efficiently;
- hint_{C_2}: a hint used to compute the deterministic C_2-torsion basis on E_A efficiently;
- $s_{A,2^a}$: the scalar used to recover the point $[m_{2^a}]Q_A$;
- s_{C_1}, label_{C_1} : the scalar and an integer used to recover the point $[m_{C_1}]S_A^1$;
- label_+: a boolean flag used to recover the difference of masked $2^a C_1$-torsion points;
- $s_{C_2}^1$, $s_{C_2}^2$, $s_{C_2}^3$, and label_{C_2}: the scalars and a boolean flag used to recover the points $[m_{C_2}]R_A^2$ and $[m_{C_2}]S_A^2$, where

$$m_{C_2} = \begin{cases} u_{A,C_2}^{-1} \bmod C_2, & \text{if } u_{A,C_2} \text{ is invertible in } \mathbb{Z}/C_1\mathbb{Z} \ , \\ v_{A,C_2}^{-1} \bmod C_2, & \text{otherwise.} \end{cases}$$

Algorithm 20 illustrates how to generate the compressed public key.

4.2.2 Ciphertext

The ciphertext involves two elliptic curves E_B and E_{AB}, and the evaluations of ϕ' at the 2^a-torsion, i.e., (P_B, Q_B) and (P_{AB}, Q_{AB}). In addition, it includes two points Y_B and X_{AB} of order D.

As with the compressed public key, we can use deterministic bases on E_B and E_{AB} to compute a linear representation of the 2^a-torsion bases:

$$\begin{pmatrix} P_B \\ Q_B \end{pmatrix} = \begin{pmatrix} s_{B,2^a} & t_{B,2^a} \\ u_{B,2^a} & v_{B,2^a} \end{pmatrix} \cdot \begin{pmatrix} U_{B,2^a} \\ V_{B,2^a} \end{pmatrix},$$
$$\begin{pmatrix} P_{AB} \\ Q_{AB} \end{pmatrix} = \begin{pmatrix} s_{AB,2^a} & t_{AB,2^a} \\ u_{AB,2^a} & v_{AB,2^a} \end{pmatrix} \cdot \begin{pmatrix} U_{AB,2^a} \\ V_{AB,2^a} \end{pmatrix},$$

where $(U_{B,2^a}, V_{B,2^a})$ and $(U_{AB,2^a}, V_{AB,2^a})$ are deterministic 2^a-torsion bases on E_B and E_{AB}, respectively.

To fully exploit the capacity of x-only projective coordinates and minimize the ciphertext size, we store the following points:

- Q_B^+: the point Q_B^+ represented by its x-coordinate, satisfying $[D]Q_B^+ = [m_{B,2^a}]Q_B$ and $[2^a]Q_B^+ = Y_B$;
- P_{AB}^+: the point P_{AB}^+ represented by its x-coordinate, satisfying $[D]P_{AB}^+ = [m_{AB,2^a}]P_{AB}$ and $[2^a]P_{AB}^+ = X_{AB}$;

where

$$m_{B,2^a} = v_{B,2^a}^{-1} \bmod 2^a,$$
$$m_{AB,2^a} = u_{AB,2^a}^{-1} \bmod 2^a.$$

Then the points $[m_{B,2^a}]P_B$ and $[m_{AB,2^a}]Q_{AB}$ can be stored via a single scalar in $\mathbb{Z}/2^a\mathbb{Z}$, respectively.

To summarize, the compressed ciphertext is of the form

$$\mathsf{ct} = (E_B, E_{AB}, Q_B^+, P_{AB}^+, \mathsf{hint}_B, \mathsf{hint}_{AB}, s_{B,2^a}, s_{AB,2^a}, c),$$

where:

- E_B: the elliptic curve E_B represented by its Montgomery coefficient;
- E_{AB}: the elliptic curve E_{AB} represented by its Montgomery coefficient;
- Q_B^+: the point Q_B^+ represented by its x-coordinate, satisfying $[D]Q_B^+ = [m_{B,2^a}]Q_B$ and $[2^a]Q_B^+ = Y_B$;
- P_{AB}^+: the point P_{AB}^+ represented by its x-coordinate, satisfying $[D]P_{AB}^+ = [m_{AB,2^a}]P_{AB}$ and $[2^a]P_{AB}^+ = X_{AB}$;
- hint_B: a hint used to compute the deterministic 2^a-torsion basis on E_B efficiently;
- hint_{AB}: a hint used to compute the deterministic 2^a-torsion basis on E_{AB} efficiently;
- $s_{B,2^a}$: the scalar used to recover the point $[m_{B,2^a}]P_B$;
- $s_{AB,2^a}$: the scalar used to recover the point $[m_{AB,2^a}]Q_{AB}$;
- c: the masked message $\mathsf{pad} \oplus \mathsf{msg}$.

Algorithms 21 and 22 introduce how to generate the compressed ciphertext and decrypt it to compute the plaintext.

Algorithm 18 QIMEN-PIKE.KeyEncUncompressed($\mathsf{pk}, \mathsf{msg}$)

Input: The public key $\mathsf{pk} = (x_{P_A^+}, x_{Q_A^+}, x_{R_A^-}, x_{S_A^-}, x_{T_A^-}, \mathsf{label}_{\mathsf{pk}})$ and a message $\mathsf{msg} \in \{0,1\}^{2\lambda}$

Output: The ciphertext ct

$E_A \leftarrow \text{RecoverCodomain}((x_{R_A^-} : 1), (x_{S_A^-} : 1), (x_{T_A^-} : 1))$
$r_1 \xleftarrow{\$} \mathbb{Z}/C_1\mathbb{Z}$, $r_2 \xleftarrow{\$} \mathbb{Z}/C_2\mathbb{Z}$, $\omega_2 \xleftarrow{\$} (\mathbb{Z}/2^a\mathbb{Z})^\times$, $\eta_D, \zeta_D \xleftarrow{\$} (\mathbb{Z}/D\mathbb{Z})^\times$
$w' \leftarrow w_0^{\eta_D \zeta_D}$
$K_1 \leftarrow \text{Ladder3pt}(R_0^1, S_0^1, R_0^1 - S_0^1, E_0, r_1)$
$K_2 \leftarrow \text{Ladder3pt}(R_0^2, S_0^2, R_0^2 - S_0^2, E_0, r_2)$
$\mathsf{pad} \leftarrow \mathsf{H}(\mathsf{KDF} \| \mathrm{tr}(w'))$
$E'_B, (T_1, T_2, K'_2) \leftarrow \text{OddIsogenyChain}(E_0, K_1, C_1, (P_0, Q_0^+, K_2))$
$E_B, (T_1, T_2) \leftarrow \text{OddIsogenyChain}(E'_B, K'_2, C_2, (T_1, T_2))$
Compute s such that $s \bmod 2^a \equiv \omega_2^{-1}$ and $s \bmod D \equiv \eta_D$
$P_B^+ \leftarrow [\omega_2]T_1$, $Q_B^+ \leftarrow [s]T_2$
$(x_{[D]P_A^+} : z_{[D]P_A^+}) \leftarrow \text{Ladder}((x_{P_A^+} : 1), E_A, D)$
$T_3 \leftarrow \text{ProjectiveDifference}((x_{[D]P_A^+} : z_{[D]P_A^+}), (x_{Q_A^+} : 1), E_A)$
if $\mathsf{label}_{\mathsf{pk}} = 0$ **then**
    $T_3 \leftarrow \text{xADD}((x_{[D]P_A^+} : z_{[D]P_A^+}), (x_{Q_A^+} : 1), T_3)$
$K_1 \leftarrow \text{Ladder3pt}([D]P_A^+, Q_A^+, T_3, E_A, r_1)$
$K_1 \leftarrow [2^a]K_1$
$K_2 \leftarrow \text{Ladder3pt}((x_{R_A^-} : 1), (x_{S_A^-} : 1), (x_{T_A^-} : 1), E_A, r_2)$
$E'_{AB}, (T_1, T_2, K'_2) \leftarrow \text{OddIsogenyChain}(E_A, K_1, C_1, (P_A^+, Q_A^+, K_2))$
$E_{AB}, (T_1, T_2) \leftarrow \text{OddIsogenyChain}(E'_{AB}, K'_2, C_2, (T_1, T_2))$
Compute s such that $s \bmod 2^a \equiv \omega_2$ and $s \bmod D \equiv \zeta_D$
$P_{AB}^+ \leftarrow [s]T_1$, $Q_{AB}^+ \leftarrow [\omega_2^{-1} \bmod 2^a]T_2$
$c \leftarrow \mathsf{pad} \oplus \mathsf{msg}$
$\mathsf{ct} \leftarrow (E_B, E_{AB}, x(P_B^+)/z(P_B^+), x(Q_B^+)/z(Q_B^+), x(P_{AB}^+)/z(P_{AB}^+), x(Q_{AB}^+)/z(Q_{AB}^+), c)$
return ct

Algorithm 19 QIMEN-PIKE.KeyDecUncompressed(ct, sk)

Input: The normalized ciphertext $(E_B, E_{AB}, P_B^+, Q_B^+, P_{AB}^+, Q_{AB}^+, c)$ and the secret key $\mathsf{sk} = (q, \alpha_2, \beta_2, \delta_D)$

Output: The message msg

1: $Q_B \leftarrow [D]Q_B^+$
2: $Y_B \leftarrow [2^a]Q_B^+$
3: $P_{AB} \leftarrow [D]P_{AB}^+$
4: $X_{AB} \leftarrow [2^a]P_{AB}^+$
5: $\widetilde{P_{AB}} \leftarrow [(-q\alpha_2)^{-1} \mod 2^a]P_{AB}, \widetilde{Q_{AB}} \leftarrow [(-q\beta_2)^{-1} \mod 2^a]Q_{AB}$
6: $E_M, \_, (Y_M, \_), (X_M, \_)$
$\leftarrow$ Isogeny22Chain$((P_B, \widetilde{P_{AB}}), (Q_B, \widetilde{Q_{AB}}), [(Y_B, 0_{E_B}), (0_{E_{AB}}, X_{AB})])$
7: $t \leftarrow \left(q(2^{a-2} - q)C\delta_D\right)^{-1} \bmod D$
8: $w \leftarrow$ TateOdd$(E_M, D, X_M, Y_M, X_M - Y_M)$
9: $w' \leftarrow w^t$
10: $\widetilde{\mathsf{pad}} \leftarrow \mathsf{H}(\mathsf{KDF} \| \mathrm{tr}(w'))$
11: **return** $\widetilde{\mathsf{pad}} \oplus c$

Algorithm 20 QIMEN-PIKE.KeyGenCompressed(λ)

Input: The internal security parameter λ
Output: The secret key and the public key $(\mathsf{sk}, \mathsf{pk})$

$(q, E_A, \widetilde{P_A^+}, \widetilde{Q_A^+}, T_A^+, R_A^-, S_A^-, T_A^-) \leftarrow$ QIMEN-PIKE.ModifiedCoreKeyGenIso()
$\alpha_2, \beta_2 \xleftarrow{\$} (\mathbb{Z}/2^{a-2}\mathbb{Z})^\times$, $\gamma_1 \xleftarrow{\$} (\mathbb{Z}/C_1\mathbb{Z})^\times$, $\gamma_2 \xleftarrow{\$} (\mathbb{Z}/C_2\mathbb{Z})^\times$, and $\delta_D \xleftarrow{\$} (\mathbb{Z}/D\mathbb{Z})^\times$
$U_{A,C_2}, V_{A,C_2}, \mathsf{hint}_{C_2} \leftarrow$TorsionBasisGivenOrderToHint$(E_A, C_2, \mathsf{false})$
$s_{A,C_2}, t_{A,C_2}, u_{A,C_2}, v_{A,C_2} \leftarrow$ DlogOdd$(E_A, C_2, U_{A,C_2}, V_{A,C_2}, U_{A,C_2} - V_{A,C_2}, R_A^-, S_A^-)$
if s_{A,C_2} is invertible in $(\mathbb{Z}/C_2\mathbb{Z})^*$ **then**
    $\mathsf{label}_{C_2} \leftarrow 0$, $s_{C_2}^1 \leftarrow (s_{A,C_2})^{-1} t_{A,C_2}$, $s_{C_2}^2 \leftarrow (s_{A,C_2})^{-1} u_{A,C_2}$, $s_{C_2}^3 \leftarrow (s_{A,C_2})^{-1} v_{A,C_2}$
else
    $\mathsf{label}_{C_2} \leftarrow 1$, $s_{C_2}^1 \leftarrow (t_{A,C_2})^{-1} s_{A,C_2}$, $s_{C_2}^2 \leftarrow (t_{A,C_2})^{-1} u_{A,C_2}$, $s_{C_2}^3 \leftarrow (t_{A,C_2})^{-1} v_{A,C_2}$
Compute t such that $t \equiv \beta_2 \bmod 2^a$ and $t \equiv \gamma_1 \bmod C_1$
$U_{A,2^aC_1}, V_{A,2^aC_1}, \mathsf{hint}_{C_1} \leftarrow$TorsionBasisGivenOrderToHint$(E_A, 2^aC_1, \mathsf{true})$
$U_{A,2^a} \leftarrow [C_1]U_{A,2^aC_1}$, $U_{A,C_1} \leftarrow [2^a]U_{A,2^aC_1}$, $V_{A,2^a} \leftarrow [C_1]V_{A,2^aC_1}$, $V_{A,C_1} \leftarrow [2^a]V_{A,2^aC_1}$
$u_{A,2^a}, v_{A,2^a} \leftarrow$ Dlog2Single$(E_A, a, U_{A,2^a}, V_{A,2^a}, U_{A,2^a} - V_{A,2^a}, [C_1 t]\widetilde{Q_A^+})$
$u_{A,C_1}, v_{A,C_1} \leftarrow$ DlogOddSingle$(E_A, C_1, U_{A,C_1}, V_{A,C_1}, U_{A,C_1} - V_{A,C_1}, [2^a t]\widetilde{Q_A^+})$
$m_{2^a} \leftarrow (v_{A,2^a})^{-1} \bmod 2^a$, $s_{A,2^a} \leftarrow m_{2^a} \cdot u_{A,2^a} \bmod 2^a$
$m_{C_1}, s_{C_1}, \mathsf{label}_{C_1} \leftarrow$LabelComputation$(C_1, u_{A,C_1}, v_{A,C_1})$
Compute t_1 such that $t_1 \equiv \alpha_2 m_{2^a} \bmod 2^a$, $t_1 \equiv \gamma_1 m_{C_1} \bmod C_1$ and $t_1 \equiv \delta_D \bmod D$
$P_A^+ \leftarrow [t_2]\widetilde{P_A^+}$
Compute t_2 such that $t_2 \equiv \beta_2 m_{2^a} \bmod 2^a$ and $t_2 \equiv \beta_2 m_{C_1} \bmod C_1$
$T_1 \leftarrow$ LadderBiscalar$(P_A^+, Q_A^+, T_A^+, E_A, Dt_1, t_2)$
$P_A^+ \leftarrow [t_1]P_A^+$, $Q_A^+ \leftarrow [t_2]Q_A^+$
$T_2 \leftarrow$ ProjectiveDifference$([D]P_A^+, Q_A^+, E_A)$
$\mathsf{label}_+ \leftarrow (T_1 = T_2)$
$\mathsf{sk} \leftarrow (q, \alpha_2, \beta_2, \delta_D)$
$\mathsf{pk} \leftarrow (E_A, P_A^+, \mathsf{hint}_{C_1}, \mathsf{hint}_{C_2}, s_{A,2^a}, s_{C_1}, \mathsf{label}_{C_1}, \mathsf{label}_+, s_{C_2}^1, s_{C_2}^2, s_{C_2}^3, \mathsf{label}_{C_2})$
return $(\mathsf{sk}, \mathsf{pk})$

Algorithm 21 QIMEN-PIKE.KeyEncCompressed($\mathsf{pk}, \mathsf{msg}$)

Input: The public key $\mathsf{pk} = (E_A, P_A^+, \mathsf{hint}_{C_1}, \mathsf{hint}_{C_2}, s_{A,2^a}, \mathsf{label}_{2^a}, s_{C_1}, \mathsf{label}_{C_1}, \mathsf{label}_+, s_{C_2}^1, s_{C_2}^2, s_{C_2}^3, \mathsf{label}_{C_2})$ and a message $\mathsf{msg} \in \{0,1\}^{2\lambda}$
Output: The ciphertext ct

$r_1 \xleftarrow{\$} \mathbb{Z}/C_1\mathbb{Z}$, $r_2 \xleftarrow{\$} \mathbb{Z}/C_2\mathbb{Z}$, $\omega_2 \xleftarrow{\$} (\mathbb{Z}/2^a\mathbb{Z})^\times$, $\eta_D, \zeta_D \xleftarrow{\$} (\mathbb{Z}/D\mathbb{Z})^\times$
$w' \leftarrow w_0^{\eta_D \zeta_D}$
$\mathsf{pad} \leftarrow \mathsf{H}(\mathsf{KDF} \| \mathrm{tr}(w'))$
$K_1 \leftarrow \textsc{Ladder3pt}(R_0^1, S_0^1, R_0^1 - S_0^1, E_0, r_1)$
$K_2 \leftarrow \textsc{Ladder3pt}(R_0^2, S_0^2, R_0^2 - S_0^2, E_0, r_2)$
$E'_B, (T_1, T_2, K'_2) \leftarrow \textsc{OddIsogenyChain}(E_0, K_1, C_1, (P_0, Q_0^+, K_2))$
$E_B, (T_1, T_2) \leftarrow \textsc{OddIsogenyChain}(E'_B, K'_2, C_2, (T_1, T_2))$
$U_{B,2^a}, V_{B,2^a}, \mathsf{hint}_B \leftarrow \textsc{TorsionBasisToHint}(E_B, a)$
$u_{B,2^a}, v_{B,2^a} \leftarrow \textsc{Dlog2Single}(E_B, a, U_{B,2^a}, V_{B,2^a}, [\omega_2]T_1)$
Compute s such that $s \bmod 2^a \equiv (\omega_2 v_{B,2^a})^{-1}$ and $s \bmod D \equiv \eta_D$
$Q_B^+ \leftarrow [s]T_2$
$U_{A,2^aC_1}, V_{A,2^aC_1} \leftarrow \textsc{TorsionBasisGivenOrderFromHint}(E_A, 2^aC_1, \mathsf{hint}_{C_1})$
$U_{A,C_2}, V_{A,C_2} \leftarrow \textsc{TorsionBasisGivenOrderFromHint}(E_A, C_2, \mathsf{hint}_{C_2})$
$S_A^- \leftarrow \textsc{LadderBiscalar}(U_{A,C_2}, V_{A,C_2}, U_{A,C_2} - V_{A,C_2}, E_A, s_{C_2}^2, s_{C_3}^3)$
if $\mathsf{label}_{C_2} = 1$ **then**
    $R_A^- \leftarrow \textsc{Ladder3pt}(V_{A,C_2}, U_{A,C_2}, U_{A,C_2} - V_{A,C_2}, E_A, s_{C_2}^1)$
    $T_A^- \leftarrow \textsc{LadderBiscalar}(U_{A,C_2}, V_{A,C_2}, U_{A,C_2} - V_{A,C_2}, E_A, s_{C_2}^1 - s_{C_2}^2, 1 - s_{C_2}^3)$
else
    $R_A^- \leftarrow \textsc{Ladder3pt}(U_{A,C_2}, V_{A,C_2}, U_{A,C_2} - V_{A,C_2}, E_A, s_{C_2}^1)$
    $T_A^- \leftarrow \textsc{LadderBiscalar}(U_{A,C_2}, V_{A,C_2}, U_{A,C_2} - V_{A,C_2}, E_A, 1 - s_{C_2}^2, s_{C_2}^1 - s_{C_2}^3)$
$t_1, t_2 \leftarrow \textsc{ScalarRecoveryFromLabel}(C_1, s_{C_1}, \mathsf{label}_{C_1})$
Compute s_1 such that $s_1 \equiv s_{A,2^a} \bmod 2^a$ and $s_1 \equiv t_1 \bmod C_1$
Compute s_2 such that $s_2 \equiv 1 \bmod 2^a$ and $s_2 \equiv t_2 \bmod C_1$
$Q_A^+ \leftarrow \textsc{LadderBiscalar}(U_{A,2^aC_1}, V_{A,2^aC_1}, U_{A,2^aC_1} - V_{A,2^aC_1}, E_A, s_1, s_2)$
$T_3 \leftarrow \textsc{ProjectiveDifference}([D]P_A^+, Q_A^+, E_A)$
if $\mathsf{label}_{\mathsf{pk}} = 0$ **then**
    $T_3 \leftarrow \textsc{xADD}([D]P_A^+, Q_A^+, T_3)$
$K_1 \leftarrow \textsc{Ladder3pt}([D]P_A^+, Q_A^+, T_3, E_A, r_1)$
$K_1 \leftarrow [2^a]K_1$
$K_2 \leftarrow \textsc{Ladder3pt}(R_A^-, S_A^-, T_A^-, E_A, r_2)$
$E'_{AB}, (T_1, T_2, K'_2) \leftarrow \textsc{OddIsogenyChain}(E_A, K_1, C_1, ([C_1D]P_A^+, [C_1]Q_A^+, K_2))$
$E_{AB}, (T_1, T_2) \leftarrow \textsc{OddIsogenyChain}(E'_{AB}, K'_2, C_2, (T_1, T_2))$
$U_{AB,2^a}, V_{AB,2^a}, \mathsf{hint}_{AB} \leftarrow \textsc{TorsionBasisToHint}(E_{AB}, a)$
$u_{AB,2^a}, v_{AB,2^a} \leftarrow \textsc{Dlog2Single}(E_{AB}, a, U_{AB,2^a}, V_{AB,2^a}, [\omega_2^{-1} \bmod 2^a]T_2)$
Compute s such that $s \bmod 2^a \equiv \omega_2 (u_{AB,2^a})^{-1}$ and $s \bmod D \equiv \zeta_D$
$P_{AB}^+ \leftarrow [s]T_1$
$s_{B,2^a} \leftarrow (v_{B,2^a})^{-1} u_{B,2^a} \bmod 2^a$, $s_{AB,2^a} \leftarrow (u_{AB,2^a})^{-1} v_{AB,2^a} \bmod 2^a$
$c \leftarrow \mathsf{pad} \oplus \mathsf{msg}$
$\mathsf{ct} = (E_B, E_{AB}, Q_B^+, P_{AB}^+, \mathsf{hint}_B, \mathsf{hint}_{AB}, s_{B,2^a}, s_{AB,2^a}, c)$
return ct

Algorithm 22 QIMEN-PIKE.KeyDecCompressed(ct, sk)

Input: The ciphertext $\mathsf{ct} = (E_B, E_{AB}, Q_B^+, P_{AB}^+, \mathsf{hint}_B, \mathsf{hint}_{AB}, s_{B,2^a}, s_{AB,2^a}, c)$ and the secret key $\mathsf{sk} = (q, \alpha_2, \beta_2, \delta_D)$

Output: The message msg

$U_{B,2^a}, V_{B,2^a} \leftarrow$ TorsionBasisFromHint$(E_B, a, \mathsf{hint}_B)$
$U_{AB,2^a}, V_{AB,2^a} \leftarrow$ TorsionBasisFromHint$(E_{AB}, a, \mathsf{hint}_{AB})$
$Q_B \leftarrow [D]Q_B^+$
$Y_B \leftarrow [2^a]Q_B^+$
$P_{AB} \leftarrow [D]P_{AB}^+$
$X_{AB} \leftarrow [2^a]P_{AB}^+$
$P_B \leftarrow$ Ladder3pt$(V_{B,2^a}, U_{B,2^a}, V_{B,2^a} - U_{B,2^a}, E_B, s_{B,2^a})$
$Q_{AB} \leftarrow$ Ladder3pt$(U_{AB,2^a}, V_{AB,2^a}, U_{AB,2^a} - V_{AB,2^a}, E_{AB}, s_{AB,2^a})$
$\widetilde{P_{AB}} \leftarrow [(-q\alpha_2)^{-1} \mod 2^a]P_{AB}, \widetilde{Q_{AB}} \leftarrow [(-q\beta_2)^{-1} \mod 2^a]Q_{AB}$
$E_M, \_, (Y_M, \_), (X_M, \_)$
$\quad \leftarrow$ Isogeny22Chain$((P_B, \widetilde{P_{AB}}), (Q_B, \widetilde{Q_{AB}}), [(Y_B, 0_{E_{AB}}), (0_{E_B}, X_{AB})])$
$t \leftarrow \left(q(2^{a-2} - q)C\delta_D\right)^{-1} \mod D$
$w \leftarrow$ TateOdd$(E_M, D, X_M, Y_M, X_M - Y_M)$
$w' \leftarrow w^t$
$\widetilde{\mathsf{pad}} \leftarrow \mathsf{H}(\mathsf{KDF} \| \mathrm{tr}(w'))$
return $\mathsf{pad} \oplus c$

CHAPTER 5

Parameter sets

This chapter specifies the concrete parameter sets supported by QIMEN-PIKE. A parameter set fixes the scalar tuple

$$\mathsf{par} = (\lambda, p, a, C_1, C_2, D), \qquad C := C_1 C_2.$$

The internal parameter λ is set to $2\lambda_q$, where λ_q denotes the target quantum security level. Together with the base curve and torsion bases of Section 3.1, these values determine the public parameters used by the scheme. The normative requirements on these parameters are given in Section 3.1.1. In this chapter, we instantiate those requirements with concrete values and record the corresponding serialization length

$$\ell_p := \left\lceil \frac{\log_2 p}{8} \right\rceil,$$

so that an element of $\mathbb{F}_p$ occupies ℓ_p bytes and an element of $\mathbb{F}_{p^2}$ occupies $2\ell_p$ bytes in a minimal byte encoding. For the implementation-level size counts reported in Table 7.1, we use the encoded slot length B_p^{enc} for an $\mathbb{F}_p$ element, corresponding to the implementation constant `FP_ENCODED_BYTES`, where

$$B_p^{\mathrm{enc}} \in \{64, 96, 192\}$$

for NGCC-1, NGCC-2, and NGCC-3, respectively. Thus, for NGCC-1, a mathematical field element needs 60 bytes in a minimal encoding, while serialized layouts reserve a 64-byte slot. All hexadecimal constants below are interpreted as non-negative integers in base 16.

5.1 Concrete parameter sets

The exact constants are as follows.

NGCC-1.

$$\begin{aligned}
\lambda &= 160, \qquad \lceil \log_2 p \rceil = 479, \qquad a = 162, \\
C_1 &= 3^5 \cdot 7^{46}, \qquad C_2 = 11^{55}, \qquad C = C_1 C_2, \\
D &= \texttt{0xd98f6625d385bce9f7e4d9ce6ce2649712e33860f465058f}, \\
p &= 2^{162} \cdot 3^5 \cdot 7^{46} \cdot D - 1,
\end{aligned}$$

NGCC-2.

$$\begin{aligned}
\lambda &= 256, \qquad \lceil \log_2 p \rceil = 765, \qquad a = 260, \\
C_1 &= 3 \cdot 5^{69}, \qquad C_2 = 7^{125}, \qquad C = C_1 C_2, \\
D &= \texttt{0x5239d20d58a01897cbfcc9d63b3cf9e34eeb43028119538d9503e} \\
&\quad \texttt{9c952d48d37a675ad7}, \\
p &= 2^{260} \cdot 3 \cdot 5^{69} \cdot D \cdot \texttt{0x190CA2A8D6DF6A79} - 1,
\end{aligned}$$

NGCC-3.

$$\begin{aligned}
\lambda &= 512, \qquad \lceil \log_2 p \rceil = 1534, \qquad a = 514, \\
C_1 &= 5 \cdot 7^{60}, \qquad C_2 = 11^{247}, \qquad C = C_1 C_2, \\
D &= \texttt{0x7827d2cf7534a7becfcfdf39f6058f7ed5c272e8e3a5fc87d9995} \\
&\quad \texttt{1e3af5445c77eea9c23e06ea7ff55ecb2124dcefad44cf3a4f361aa} \\
&\quad \texttt{d70f76964cb21884adc186a2c84a1a64414262c221f79b5c80defe6} \\
&\quad \texttt{9d80a585c0ba638e935d3bd2620299059658e862b43ef}, \\
p &= 2^{514} \cdot 5 \cdot 7^{60} \cdot D \cdot \texttt{0x2B271} - 1.
\end{aligned}$$

CHAPTER 6

Test vectors

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

For QIMEN-PIKE, the files are `KAT_KEM_NGCC-1.txt`, `KAT_KEM_NGCC-2.txt`, and `KAT_KEM_NGCC-3.txt`; each record contains a deterministic seed, a public key, a secret key, a ciphertext, and the resulting shared secret. 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 PIKE 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-PIKE. These are:

- `hd`: module to compute $(2, 2)$-isogenies in the theta model.
- `klpt`: KLPT [KLPT14] module. In particular, we use this module to generate an endomorphism in key generation.
- `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_COMPRESSED=ON` builds the compressed PIKE implementation used by the NGCC KAT generation programs.
- `PIKE_NGCC_XOF_BACKEND` selects the NGCC XOF backend; supported values are `shake` and `sm3`.
- `ENABLE_NGCC_PIKE=ON` builds the NGCC KAT generation targets. The KAT output directory is selected by `NGCC_PIKE_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.

[1] https://github.com/Kaizhan-Lin/PIKE-C-Implementation

7.1 Key and ciphertext sizes

Table 7.1 lists the public key, secret key, and ciphertext sizes per parameter set. The PIKE sizes use the encoded slot lengths $B_p^{\text{enc}} = 64, 96, 192$ for NGCC-1, NGCC-2, and NGCC-3, respectively.

Table 7.1: QIMEN-PIKE key and ciphertext sizes in bytes. Here pk, ct, and sk denote public key, ciphertext, and secret key, respectively.

Parameter set	**Uncompressed**			**Compressed**		
	pk	ct	sk	pk	ct	sk
NGCC-1	641 B	800 B	724 B	405 B	602 B	488 B
NGCC-2	961 B	1184 B	1103 B	595 B	882 B	737 B
NGCC-3	1921 B	2368 B	2220 B	1201 B	1746 B	1500 B

7.2 Performance evaluation

This section gives the performance of both the reference implementation and optimized implementation of QIMEN-PIKE 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 `PIKE_NGCC_XOF_BACKEND=sm3`, which sets `PIKE_XOF_BACKEND=2`.
- **Compilation settings:** the effective C flags included `-O3`, `-DNDEBUG`, `-std=c99`, `-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 KEM operation is averaged over 300 runs.

7.2.1 Reference Implementation

Table 7.2 shows the reference implementation performance.

Table 7.2: Reference QIMEN-PIKE KEM performance with uncompressed and compressed variants 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	**Uncompressed**			**Compressed**		
	KeyGen	Encaps	Decaps	KeyGen	Encaps	Decaps
NGCC-1	94.15	34.73	61.43	159.61	61.27	98.02
NGCC-2	350.45	121.33	222.20	608.92	223.68	360.88
NGCC-3	3005.44	1029.46	1794.11	4606.19	1803.16	2845.02

Table 7.3: Reference QIMEN-PIKE KEM throughput with uncompressed and compressed variants 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	**Uncompressed**			**Compressed**		
	KeyGen	**Encaps**	**Decaps**	**KeyGen**	**Encaps**	**Decaps**
NGCC-1	30.4	84.7	47.5	17.4	47.5	29.5
NGCC-2	7.9	22.0	12.8	4.5	13.0	8.1
NGCC-3	0.9	2.62	1.48	0.62	1.47	0.97

7.2.2 Optimized Implementation

The optimized build used

`PIKE_BUILD_TYPE=ref-mbmi2-madx`,

enabling x86-64 BMI2/ADX assembly field arithmetic. Table 7.4 show the assembly-optimized implementation performance on the platform.

Table 7.4: Assembly-optimized QIMEN-PIKE KEM performance with uncompressed and compressed variants 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	**Uncompressed**			**Compressed**		
	KeyGen	Encaps	Decaps	KeyGen	Encaps	Decaps
NGCC-1	76.16	26.94	47.93	129.31	48.00	77.45
NGCC-2	292.40	93.86	171.85	481.88	172.94	280.08
NGCC-3	2480.87	803.12	1405.36	3639.78	1421.09	2243.85

Table 7.5: Optimized QIMEN-PIKE KEM throughput with uncompressed and compressed variants 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	**Uncompressed**			**Compressed**		
	KeyGen	**Encaps**	**Decaps**	**KeyGen**	**Encaps**	**Decaps**
NGCC-1	34.9	104.2	54.7	20.7	55.0	36.8
NGCC-2	10.5	32.8	17.8	5.6	17.0	10.8
NGCC-3	1.08	3.78	2.15	0.83	2.07	1.31

CHAPTER 8

Security

This chapter discusses the main security aspects of QIMEN-PIKE, following the analyses conducted in [CCLL26] and [BM25]. First, Section 8.1 discusses the complexity of the endomorphism ring problem, a central hardness assumption underlying the security of supersingular isogeny-based cryptography. Then, in Section 8.2, we state that QIMEN-PIKE is IND-CCA2, under the hardness of Theorem 8.2.1, in the quantum random-oracle model and briefly discuss the relation between Theorem 8.2.1 and the security of POKÉ [BM25]. Finally, in Section 8.3, we examine the complexity of different concrete attacks on QIMEN-PIKE. In Section 8.4, we summarize the attacks and justify that our parameter choices meet the required classical and quantum security levels specified by NGCC.

8.1 The endomorphism ring problem

The endomorphism ring problem is the common foundation for all of (supersingular) isogeny-based cryptography, including QIMEN-PIKE, and consequently its hardness forms an upper bound for the hardness of breaking this submission. As discussed in Section 2.6.2 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.

8.2 Theoretical Security

The post-quantum secure KEM QIMEN-PIKE is obtained by first constructing QIMEN-PIKE.PKE, an IND-CPA PKE, following the ElGamal paradigm, see Section 3.2, and then applying the Fujisaki-Okamoto transform [FO99] to obtain QIMEN-PIKE.KEM, an IND-CCA2 KEM, see Section 3.3.

Hence, to prove the security of QIMEN-PIKE, we identify the hard problem underlying the IND-CPA security of QIMEN-PIKE.PKE and derive the IND-CCA2 security of QIMEN-PIKE.KEM, under the hardness of this problem, using classical results on the Fujisaki-Okamoto transform. This central problem, which we call Computational PIKE Problem, is defined as follows.

Problem 8.2.1 (Computational PIKE Problem [CCLL26, Problem 9])**.** The problem is parameterized by pp. $\mathsf{pp} = (p, a, C, D, E_0, (P_0, Q_0), (R_0, S_0), (X_0, Y_0))$ where E_0 is a supersingular elliptic curve defined over $\mathbb{F}_{p^2}$ with known endomorphism ring, $\{P_0, Q_0\}$, $\{R_0, S_0\}$ and $\{X_0, Y_0\}$ are the bases of $E_0[2^a]$, $E_0[C]$ and $E_0[D]$ respectively.

The challenger first randomly generates an isogeny $\phi : E_0 \to E_A$ of degree $q(2^{a-2} - q)$ for some unknown value $q < 2^{a-2}$ coprime to 2, C and D, then computes $\mathsf{x}_1 = (E_A, P_A, Q_A, R_A, S_A, X_A)$ where

$$P_A, Q_A, R_A, S_A, X_A = \phi([\alpha_2]P_0,\ [\beta_2]Q_0,\ [\gamma_C]R_0,\ [\gamma_C]S_0,\ [\delta_D]X_0)$$

and $\alpha_2, \beta_2, \gamma_C, \delta_D$ are sampled uniformly at random from $(\mathbb{Z}/n\mathbb{Z})^\times$ for $n = 2^a, 2^a, C, D$, respectively.

Then, the challenger generates an isogeny $\psi : E_0 \to E_B$ of degree C randomly, and writes $\psi' : E_A \to E_{AB}$ for its pushforward $[\phi]_*\psi$. Then, the challenger computes $\mathsf{x}_2 = (P_B, Q_B, Y_B, P_{AB}, Q_{AB}, X_{AB})$ defined as follows:

$$\begin{aligned} &P_B, Q_B, Y_B, P_{AB}, Q_{AB}, X_{AB} \\ &= \psi([\omega_2]P_0,\ [\omega_2^{-1}]Q_0,\ [\eta_D]Y_0),\ \psi'([\omega_2]P_A,\ [\omega_2^{-1}]Q_A,\ [\zeta_D]X_A), \end{aligned}$$

where $\omega_2, \eta_D, \zeta_D$ are drawn uniformly random from $(\mathbb{Z}/n\mathbb{Z})^\times$ for $n = 2^a, D, D$ respectively.

Given $\mathsf{pp}, \mathsf{x}_1, \mathsf{x}_2$, compute $e(X_0, Y_0)^{\eta_D \zeta_D}$.

We denote the advantage of the adversary $\mathcal{A}$ by $\mathsf{Adv}^{\mathsf{CPIKE}}_{\mathsf{pp}}(\mathcal{A}) = \Pr[\mathcal{A} \text{ win.}]$ where the probability is taken over the randomness used in the game and by $\mathcal{A}$.

8.2.1 IND-CPA Security of QIMEN-PIKE.PKE

The correctness of QIMEN-PIKE.PKE is first established by [CCLL26, Proposition 8], after which the IND-CPA security of the scheme is proven under Theorem 8.2.1.

Theorem 8.2.2 ([CCLL26, Theorem 10])**.** *If Theorem 8.2.1 is hard, then the PKE scheme* Π_{PKE} *described in Chapter 3 is IND-CPA by modeling* H *as a random oracle. Precisely, given an adversary* $\mathcal{A}$ *against IND-CPA of* Π_{PKE} *with advantage* ϵ*,* Q *queries to the random oracle and running in time* T*, we can construct an adversary* $\mathcal{B}$ *running in time* $O(T)$ *against Theorem 8.2.1 such that*

$$\mathsf{Adv}^{\mathsf{IND\text{-}CPA}}_{\Pi_{\mathsf{PKE}},\mathsf{pp}}(\mathcal{A}) \leq 2Q \cdot \mathsf{Adv}^{\mathsf{CPIKE}}_{\mathsf{pp}}(\mathcal{B}).$$

8.2.2 IND-CCA2 Security of PIKE.KEM

The results of [HHK17, JZM19, DFMS22] are sufficient to show the IND-CCA2 security of $\mathsf{QIMEN-PIKE.CCAKEM}$ from the IND-CPA security of QIMEN-PIKE.PKE in either the classical random oracle model or the quantum random oracle model. The formal statements are as follows.

Theorem 8.2.3 (IND-CCA2 Security of PIKE.KEM)**.** *Let* $\mathcal{M}$ *denote the message space of QIMEN-PIKE.PKE. If QIMEN-PIKE.PKE is IND-CPA secure, then QIMEN-PIKE.KEM is IND-CCA2 secure in the classical ROM and the QROM. Concretely, for an arbitrary IND-CCA adversary* $\mathcal{A}$ *against QIMEN-PIKE.KEM with at most* q_G *queries to the random oracle* G *and at most* q_H *queries to the random oracles* H_{ct} *and* KDF *in total, we have that*

(1) *in the classical random-oracle model [HHK17] there exists an IND-CPA adversary* $\mathcal{B}_{\mathrm{cROM}}$ *against QIMEN-PIKE.PKE such that*

$$\mathsf{Adv}^{\mathsf{IND\text{-}CCA2}}_{\mathit{QIMEN\text{-}PIKE.KEM},\mathsf{pp}}(\mathcal{A}) \leq 3\mathsf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathit{QIMEN\text{-}PIKE.PKE},\mathsf{pp}}(\mathcal{B}_{\mathrm{cROM}}) + \Delta_{\mathrm{cROM}}(\mathcal{A}),$$

where

$$\Delta_{\mathrm{cROM}}(\mathcal{A}) := \frac{2q_G + q_H + 1}{|\mathcal{M}|}.$$

(2) *in the quantum random-oracle model [JZM19] there exists an IND-CPA adversary* $\mathcal{B}_{\mathrm{qROM}}$ *against QIMEN-PIKE.PKE such that*

$$\mathsf{Adv}^{\mathsf{IND\text{-}CCA2}}_{\mathit{QIMEN\text{-}PIKE.KEM},\mathsf{pp}}(\mathcal{A}) \leq 2\sqrt{q_G + q_H + 1}\,\mathsf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathit{QIMEN\text{-}PIKE.PKE},\mathsf{pp}}(\mathcal{B}_{\mathrm{qROM}}) + \Delta_{\mathrm{qROM}}(\mathcal{A}),$$

where

$$\Delta_{\mathrm{qROM}}(\mathcal{A}) := \frac{2q_H}{\sqrt{|\mathcal{M}|}} + \frac{2(q_G + q_H + 1)^2}{|\mathcal{M}|}.$$

8.2.3 On (In-)Equivalence with POKÉ

A natural question is whether there is a one-sided reduction between the Theorem 8.2.1 and the C-POKÉ problem [BM25, Problem 7] underlying POKÉ security. In [CCLL26], it is argued that the answer is negative in a strict sense, due to subtle but consequential differences in the settings. In our public key, while PIKE exposes one torsion point less, it is of larger order than POKÉ; moreover, although a higher-order torsion point is provided in PIKE, no basis evaluation is revealed as in POKÉ. Consequently, for key-recovery style attacks it is difficult to say which public key is inherently more robust.

For the ciphertext torsion data on E_B and E_{AB}, POKÉ publishes two torsion-point evaluations on E_B, whereas we publish one each on E_B and E_{AB}. These structural differences suggest that establishing a clean equivalence between the two assumptions is unlikely.

8.3 Practical security

In this section, we use the notations from Fig. 3.1.

8.3.1 Isogeny-recovery problems

Following [CCLL26], to analyze the practical security of QIMEN-PIKE, we derive from Theorem 8.2.1 two subproblems, each consisting of recovering a secret isogeny. The first problem concerns the secret isogeny of the receiver and corresponds to recovering the secret key, while the second problem concerns the secret isogeny of the sender and corresponds to recovering the nonce isogeny. Solving either of these problems is enough to break the security of QIMEN-PIKE. We remind the reader that the generic problem of finding an isogeny between two given supersingular elliptic curves is equivalent to Problem 8.1.1 [PW24, HLMW26].

Problem 8.3.1 (Receiver Isogeny Recovery Problem [CCLL26, Problem 13])**.** The problem is parameterized by the same public parameters $\mathsf{pp} = (p, a, C, D, E_0, (P_0, Q_0), (R_0, S_0), (X_0, Y_0))$ as in Theorem 8.2.1. The challenger samples an isogeny $\phi : E_0 \to E_A$ of degree $q(2^{a-2} - q)$, where $q < 2^{a-2}$ is unknown to the adversary and is coprime to 2, C, and D. It then samples masking scalars $\alpha_2, \beta_2 \xleftarrow{\$} (\mathbb{Z}/2^a\mathbb{Z})^\times$, $\gamma_C \xleftarrow{\$} (\mathbb{Z}/C\mathbb{Z})^\times$, and $\delta_D \xleftarrow{\$} (\mathbb{Z}/D\mathbb{Z})^\times$, and sets $\mathsf{x}_1 := (E_A, P_A, Q_A, R_A, S_A, X_A)$ with

$$\begin{pmatrix} P_A \\ Q_A \end{pmatrix} = \begin{pmatrix} \alpha_2 & 0 \\ 0 & \beta_2 \end{pmatrix} \phi \begin{pmatrix} P_0 \\ Q_0 \end{pmatrix}, \qquad \begin{pmatrix} R_A \\ S_A \end{pmatrix} = \begin{pmatrix} \gamma_C & 0 \\ 0 & \gamma_C \end{pmatrix} \phi \begin{pmatrix} R_0 \\ S_0 \end{pmatrix}, \qquad X_A = [\delta_D]\, \phi(X_0). \tag{8.1}$$

Given $(\mathsf{pp}, \mathsf{x}_1)$, recover (an explicit representation of) ϕ.

Problem 8.3.2 (Sender Isogeny Recovery Problem [CCLL26, Problem 14])**.** The problem is parameterized by the same public parameters pp as above. The challenger first generates

$\mathsf{x}_1 := (E_A, P_A, Q_A, R_A, S_A, X_A)$ as in Theorem 8.3.1. It then samples an isogeny $\psi : E_0 \to E_B$ of degree C uniformly at random, and lets $\psi' : E_A \to E_{AB}$ denote the pushforward $[\phi]_*\psi$. Finally, it samples $\omega_2 \xleftarrow{\$} (\mathbb{Z}/2^a\mathbb{Z})^\times$ and $\eta_D, \zeta_D \xleftarrow{\$} (\mathbb{Z}/D\mathbb{Z})^\times$, and defines $\mathsf{x}_2 := (E_B, P_B, Q_B, Y_B, E_{AB}, P_{AB}, Q_{AB}, X_{AB})$ via

$$\begin{pmatrix} P_B \\ Q_B \end{pmatrix} = \begin{pmatrix} \omega_2 & 0 \\ 0 & \omega_2^{-1} \end{pmatrix} \cdot \psi \begin{pmatrix} P_0 \\ Q_0 \end{pmatrix}, \qquad \begin{pmatrix} P_{AB} \\ Q_{AB} \end{pmatrix} = \begin{pmatrix} \omega_2 & 0 \\ 0 & \omega_2^{-1} \end{pmatrix} \cdot \psi' \begin{pmatrix} P_A \\ Q_A \end{pmatrix}, \tag{8.2}$$
$$Y_B = [\eta_D]\,\psi(Y_0), \qquad X_{AB} = [\zeta_D]\,\psi'(X_A).$$

Given $(\mathsf{pp}, \mathsf{x}_1, \mathsf{x}_2)$, recover ψ or ψ'.

Before studying their respective concrete complexity, we assess Theorems 8.3.1 and 8.3.2 against known isogeny-recovery approaches. Following [CCLL26, Section 5.2], we focus on the attacks in [CV23, DFP24, BM25]. The first line of work [CV23, DFP24] generalizes key-recovery-style attacks when an adversary is given a masked basis evaluation of a known-degree isogeny; the core observation is that certain diagonal masking matrices commute with endomorphisms and can enable reductions to SIDH-style recovery in specific settings. This does not apply to Theorem 8.3.1: in Eq. (8.1) the critical degree information about ϕ is hidden by independent masking scalars with uniformly distributed determinants, and cannot be recovered via pairings. For Theorem 8.3.2, the masking structure differs from the commutative-diagonal form exploited in these attacks.

A second line of work extends related ideas to FESTA-type settings under maliciously chosen torsion bases [CV23]. This is not applicable to the E_B and E_{AB} components in Theorem 8.3.2 under deterministic (canonical) basis generation, and moreover the transcript does not expose sufficiently many torsion points for the reduction strategy to go through.

Finally, [BM25] gives an efficient recovery attack against $(q, 2^{a-2} - q)$-isogenies under specific masking conditions; the first equation in Eq. (8.1) is not of the required form, and thus this technique does not apply to Theorem 8.3.1.

More generally, as illustrated in Fig. 1 of [DFP24], the best known generic approaches against masked-isogeny instances of the type used here still require exponential time. In conclusion, there is currently no efficient attack to recover isogenies from an individual component of the Theorem 8.2.1 transcript.

We now turn to a precise complexity analysis of applicable attacks against these problems.

8.3.2 Key-recovery (Attack on Theorem 8.3.1)

The secret key consists of α_2, β_2, δ_D, and $\deg\phi$. By exploiting the Tate pairing together with the SIDH key-recovery attack, one can show that recovering these quantities is equivalent to solving Theorem 8.3.1, i.e. computing the secret isogeny ϕ. Such attacks crucially rely on the images of torsion points under the secret isogeny ϕ to reconstruct it efficiently.

However, it is not necessary to recover every component of the secret key to recover ϕ. Furthermore, while the value γ_C is not part of the secret key, knowing it is also sufficient to recover ϕ. We now address each of these points in turn.

We assume that $C > \sqrt{\deg \phi}$. Under this assumption, knowledge of the secret value q, or equivalently $\deg \phi$, enables the adversary to recover ϕ. Indeed, since C is smooth, γ_C can be efficiently recovered from a Tate pairing computation, yielding $\phi(R_0)$ and $\phi(S_0)$. This data can then be fed into a SIDH key-recovery procedure to obtain the secret isogeny ϕ.

If the adversary obtains the masking scalar γ_C, then $\deg \phi \mod C$ can be recovered via the Tate pairing. Since q is approximately as large as $\sqrt{\deg \phi}$, the adversary can compute q by solving the quadratic equation

$$q(2^{a-2} - q) \equiv \deg \phi \mod C.$$

This implies that the adversary can recover the secret key ϕ using, once again, a SIDH key-recovery procedure.

Furthermore, in [BM25, Section 6.1], the authors make the observation that it is possible to recover ϕ by representing it via the 2-dimensional isogeny Φ with kernel

$$\ker \Phi = \langle ([-\alpha_2 q]P_0, [\alpha_2]\phi(P_0)), ([-\alpha_2 q]Q_0, [\alpha_2]\phi(Q_0)) \rangle.$$

When $\alpha_2 = \beta_2$, both the points $[\alpha_2]\phi(P_0) = P_1$ and $[\alpha_2]\phi(Q_0) = Q_1$ are public and one can extract $-\alpha_2 q \bmod 2^a$ from Tate pairing computations. Hence, one can efficiently compute the kernel of Φ.

This attack is mitigated by fixing $\alpha_2 \neq \beta_2$. However, given the ratio α_2/β_2, an adversary can still perform a similar recovery. Indeed, by computing $[\alpha_2/\beta_2]Q_1 = [\alpha_2]\phi(Q_0)$, one can again obtain $-\alpha_2 q \bmod 2^a$ using the Tate pairing. Then the kernel of Φ is computable efficiently, as $-\alpha_2 q \bmod 2^a$, $P_1 = [\alpha_2]\phi(P_0)$ and $[\alpha_2/\beta_2]Q_1 = [\alpha_2]\phi(Q_0)$ are known.

In summary, recovering the secret isogeny ϕ reduces to obtaining any one of the following pieces of information: the secret value q, the masking scalar γ_C, or the ratio α_2/β_2. Let us now discuss the difficulty of computing each of these secret values to derive secure parameters.

- Regarding the secret value q, there are approximately 2^{a-2} possible candidates, which is $O(2^a)$. Hence, an adversary can recover q via a brute-force search in classical time $O(2^a)$. If quantum resources are available, this complexity can be reduced to $O(2^{a/2})$ using Grover's algorithm [Gro96].
- The masking scalar γ_C is uniformly distributed over $(\mathbb{Z}/C\mathbb{Z})^\times$ whose cardinality is approximately C. However, note that $\gamma_C \bmod C'$, where C' satisfies $C' \mid C$ and $C' \approx \sqrt{\deg \phi} \approx 2^{a-2}$, is actually needed to recover ϕ, because the C'-torsion points are only required to compute q. Therefore, by testing $O(2^a)$ candidates of $\gamma_C \bmod C'$, the adversary can compute ϕ. Thus, the complexity of this approach is $O(2^a)$ on classical computers, and $O(2^{a/2})$ on quantum computers.

- The masking scalars α_2 and β_2 are assumed to be uniformly distributed over $(\mathbb{Z}/2^a\mathbb{Z})^\times$. Consequently, their ratio α_2/β_2 is also uniformly distributed over $(\mathbb{Z}/2^a\mathbb{Z})^\times$, which forms a set of cardinality approximately 2^a; therefore, the adversary can recover the ratio α_2/β_2 by exhaustive search in classical time $O(2^a)$ and in quantum time $O(2^{a/2})$.

We conclude that recovering the secret key of PIKE requires at least $O(2^a)$ time on classical computers and $O(2^{a/2})$ time on quantum computers.

8.3.3 Nonce isogeny recovery (Attack on Theorem 8.3.2)

If the nonce isogeny ψ (or ψ') can be recovered, the shared secret can be derived accordingly. There are two principal approaches to recovering ψ:

- Recover ψ via a meet-in-the-middle attack between E_0 and E_B;
- First recover the masking nonce ω_2, and subsequently apply a SIDH key-recovery attack to obtain ψ.

In the first approach, the adversary computes two isogenies whose degrees multiply to C, originating respectively from E_0 and E_B, each of degree approximately $\sqrt{C}$. If their codomains coincide, composing them yields an isogeny of degree C between E_0 and E_B. In the ideal case where ψ is the unique such isogeny, this reconstruction succeeds immediately. In general, multiple isogenies of degree C may exist between E_0 and E_B, but finding any of them already requires $O(\sqrt{C})$ operations on a classical computer. When quantum resources are available, the complexity can be reduced to $O(C^{1/3})$ by employing a claw-finding algorithm based on quantum walks [Tan09]. This provides lower bounds on the classical and quantum costs of recovering ψ with this method.

In the second approach, the adversary performs an exhaustive search to determine the masking scalar ω_2. Observe that ω_2 is sampled uniformly at random from the group $(\mathbb{Z}/2^a\mathbb{Z})^\times$, which is of size approximately 2^a.

Consequently, the second approach requires $O(2^a)$ time by using classical computers. This complexity can be reduced via quantum computing, yielding a running time of $O(2^{a/2})$.

8.3.4 Message-recovery

The shared key of PIKE is $e_D(X_0, Y_0)^{\eta_D\zeta_D}$. Therefore, an adversary may attempt a direct guess of $e_D(X_0, Y_0)^{\eta_D\zeta_D}$.

Since η_D and ζ_D are taken uniformly at random, the shared value is also uniformly distributed in μ_D, which is a group of size D. Hence, the adversary succeeds in revealing the shared value in $O(D)$ time by classical computers, and in $O(\sqrt{D})$ time by quantum computers with Grover's algorithm.

8.4 Parameter security assessment

Tables 8.1 and 8.2 report respectively the cost of the best known classical and quantum attacks on QIMEN-PIKE, instantiated with the parameters of Chapter 5, confirming that the target security levels are achieved. We use the following notations for these tables.

- $\star$: The complexity of this attack also corresponds to the first attack presented against the nonce-isogeny-recovery problem.
- §: The exponents are rounded up, e.g. we write $2^{79.7\ldots}$ as 2^{80}.

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

		NGCC-1	**NGCC-2**	**NGCC-3**
		Parameters		
	a	162	260	514
	C_1	$3^5 \cdot 7^{46}$	$3 \cdot 5^{69}$	$5 \cdot 7^{60}$
	C_2	11^{55}	7^{125}	11^{247}
	C	C_1C_2	C_1C_2	C_1C_2
	D	0xa98840 81fc61c3 910dd29f 8b9a278f 64abd4a9 1bcbcaf	0x5239d20d58 a01897cbfcc9 d63b3cf9e34e eb4302811953 8d9503e9c952 d48d37a675ad 7	0x7827d2cf753 4a7becfcfdf39 f6058f7ed5c27 2e8e3a5fc87d9 9951e3af5445c 77eea9c23e06e a7ff55ecb2124 dcefad44cf3a4 f361aad70f769 64cb21884adc1 86a2c84a1a644 14262c221f79b 5c80defe69d80 a585c0ba638e9 35d3bd2620299 059658e862b43 ef
	D'		0x190CA2A8D6 DF6A79	0x2B271
	p	$2^{162} \cdot 3^5 \cdot 7^{46} \cdot D - 1$	$2^{260} \cdot 3 \cdot 5^{69} \cdot D \cdot D' - 1$	$2^{514} \cdot 5 \cdot 7^{60} \cdot D \cdot D' - 1$
Required classical security level		128	256	512
Classical attack	**Complexity**	**Complexity estimate**[§]		
On Problem 8.1.1	$\tilde{O}(p^{1/2})$	2^{239}	2^{382}	2^{767}
Key-recovery*	$\tilde{O}(2^a)$	2^{162}	2^{260}	2^{514}
Nonce-isogeny-recovery	$\tilde{O}(C^{1/2})$	2^{164}	2^{256}	2^{513}
Message-recovery	$\tilde{O}(D)$	2^{179}	2^{282}	2^{831}

Table 8.2: Choice of parameters and quantum attack complexities for QIMEN-PIKE

		NGCC-1	**NGCC-2**	**NGCC-3**
		Parameters		
	a	162	260	514
	C_1	$3^5 \cdot 7^{46}$	$3 \cdot 5^{69}$	$5 \cdot 7^{60}$
	C_2	11^{55}	7^{125}	11^{247}
	C	C_1C_2	C_1C_2	C_1C_2
	D	0xa98840 81fc61c3 910dd29f 8b9a278f 64abd4a9 1bcbcaf	0x5239d20d58 a01897cbfcc9 d63b3cf9e34e eb4302811953 8d9503e9c952 d48d37a675ad 7	0x7827d2cf753 4a7becfcfdf39 f6058f7ed5c27 2e8e3a5fc87d9 9951e3af5445c 77eea9c23e06e a7ff55ecb2124 dcefad44cf3a4 f361aad70f769 64cb21884adc1 86a2c84a1a644 14262c221f79b 5c80defe69d80 a585c0ba638e9 35d3bd2620299 059658e862b43 ef
	D'		0x190CA2A8D6 DF6A79	0x2B271
	p	$2^{162} \cdot 3^5 \cdot 7^{46} \cdot D - 1$	$2^{260} \cdot 3 \cdot 5^{69} \cdot D \cdot D' - 1$	$2^{514} \cdot 5 \cdot 7^{60} \cdot D \cdot D' - 1$
Required quantum security level		80	128	256
Quantum attack	**Complexity**	**Complexity estimate**[§]		
On Problem 8.1.1	$\tilde{O}(p^{1/4})$	2^{120}	2^{191}	2^{383}
Key-recovery*	$\tilde{O}(2^{a/2})$	2^{81}	2^{130}	2^{257}
Nonce-isogeny-recovery	$\tilde{O}(C^{1/3})$	2^{109}	2^{171}	2^{342}
Message-recovery	$\tilde{O}(D^{1/2})$	2^{90}	2^{141}	2^{415}

CHAPTER 9

Failure analysis

The role of this chapter is to list and analyze possible failure cases of various algorithms involved in QIMEN-PIKE. 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-PIKE, the algorithms that may raise exceptions are GeneralizedRepresentInteger and the two-dimensional isogeny-chain routines. During key generation, these exceptions may be raised inside QIMEN-PIKE.CoreKeyGenIso: the call to GeneralizedRepresentInteger may fail if its bounded search does not find a quaternion element of the target norm, and the calls to Isogeny22Chain may fail if one of the internal two-dimensional isogeny subroutines fails. Such exceptions are propagated to the caller as key-generation failures. During decryption, QIMEN-PIKE.PKE.Decrypt may similarly propagate an exception raised by Isogeny22Chain; this exception is caught by QIMEN-PIKE.Decaps, which returns the implicit-rejection fallback key derived from the secret value z and the ciphertext hash.

In the following sections, we analyze the failure probabilities of GeneralizedRepresentInteger and Isogeny22Chain. We note that the failure analysis of GeneralizedRepresentInteger follows closely the analysis in [AAA+25], with numbers adjusted to our setting. 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 GeneralizedRepresentInteger

GeneralizedRepresentInteger succeeds if the algorithm executes Line 10, i.e., if CornacchiaGeneral returns a representation of M' as a sum of two squares. Under the following Heuristic (Heuristic 9.1.1), the probability of M' being a prime congruent to 1 modulo 4 is approximately $1/(2\log M')$; this is a sufficient condition for CornacchiaGeneral to succeed and gives a conservative lower bound on the success probability. In QIMEN-PIKE, we choose $M = q(2^{a-2}-q)C$ (as specified in Chapter 5), where $q < 2^{a-2}$ in QIMEN-

PIKE.CoreKeyGenIso. For all three PIKE parameter sets, this gives $M' < 4M < p^{1.5}$ whenever $M' := 4M - p(z^2 + t^2) > 0$. Hence $\log M' \leq 1.5 \log p$, and the probability of success is conservatively lower bounded by $1/(3\log(p))$.

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.

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

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

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

9.2 Failure analysis of Isogeny22Chain*

During key generation and decryption of QIMEN-PIKE 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. If they happen during decryption, with overwhelming probability they indicate a malformed ciphertext and thus lead QIMEN-PIKE.Decaps to return the implicit-rejection fallback key.

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 $1 \leq k < e$. To bound the failure probability in key generation and decryption, we introduce the following heuristic.

Heuristic 9.2.1. The surfaces A_k encountered along the chains of $(2,2)$-isogenies computed in QIMEN-PIKE.PKE.KeyGen and QIMEN-PIKE.PKE.Decrypt 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 9.2.2. Let $E_1 \times E_2$ be a product of elliptic curves to which we apply a 2-dimensional isogeny computation during an honest execution of QIMEN-PIKE.PKE.KeyGen and QIMEN-PIKE.PKE.Decrypt. Then the trace of the product coordinate $X_1 \cdot X_2$ under the gluing kernel behaves like an independent random element of a $\mathbb{F}(p^2)$ vector space of dimension one.

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

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.

[AAB+22] Carlos Aguilar-Melchor, Nicolas Aragon, Slim Bettaieb, Loïc Bidoux, Olivier Blazy, Jean-Christophe Deneuville, Philippe Gaborit, Edoardo Persichetti, Gilles Zémor, Jurjen Bos, Arnaud Dion, Jerome Lacan, Jean-Marc Robert, and Pascal Veron. HQC. Technical report, National Institute of Standards and Technology, 2022. available at https://csrc.nist.gov/Projects/post-quantum-cryptography/round-4-submissions.

[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.

[BM25] Andrea Basso and Luciano Maino. POKÉ: A compact and efficient PKE from higher-dimensional isogenies. In *EUROCRYPT 2025, Part II*, LNCS, pages 94–123. Springer, Cham, Switzerland, June 2025.

[BMP23] Andrea Basso, Luciano Maino, and Giacomo Pope. FESTA: Fast encryption from supersingular torsion attacks. In Jian Guo and Ron Steinfeld, editors, *ASIACRYPT 2023, Part VII*, volume 14444 of *LNCS*, pages 98–126, Guangzhou, China, December 4–8, 2023. Springer, Singapore, Singapore.

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

[CCLL26] Shiping Cai, Mingjie Chen, Yi-Fu Lai, and Kaizhan Lin. Pike: Faster isogeny-based public key encryption withăpairing-assisted decryption. In Shi Bai and Edoardo Persichetti, editors, *Public-Key Cryptography – PKC 2026*, pages 58–91, Cham, 2026. Springer Nature Switzerland.

[CD23] Wouter Castryck and Thomas Decru. An efficient key recovery attack on SIDH. In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part V*, volume 14008 of *LNCS*, pages 423–447, Lyon, France, April 23–27, 2023. Springer, Cham, Switzerland.

[CH17] Craig Costello and Hüseyin Hisil. A simple and compact algorithm for SIDH with arbitrary degree isogenies. In Tsuyoshi Takagi and Thomas Peyrin, editors, *ASIACRYPT 2017, Part II*, volume 10625 of *LNCS*, pages 303–329, Hong Kong, China, December 3–7, 2017. Springer, Cham, Switzerland.

[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.

[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.

[CV23] Wouter Castryck and Frederik Vercauteren. A polynomial time attack on instances of M-SIDH and FESTA. In Jian Guo and Ron Steinfeld, editors, *ASIACRYPT 2023, Part VII*, volume 14444 of *LNCS*, pages 127–156, Guangzhou, China, December 4–8, 2023. Springer, Singapore, Singapore.

[DFMS22] Jelle Don, Serge Fehr, Christian Majenz, and Christian Schaffner. Online-extractability in the quantum random-oracle model. In Orr Dunkelman and Stefan Dziembowski, editors, *EUROCRYPT 2022, Part III*, volume 13277 of *LNCS*, pages 677–706, Trondheim, Norway, May 30 – June 3, 2022. Springer, Cham, Switzerland.

[DFP24] Luca De Feo, Tako Boris Fouotsa, and Lorenz Panny. Isogeny problems with level structure. In Marc Joye and Gregor Leander, editors, *EUROCRYPT 2024, Part VII*, volume 14657 of *LNCS*, pages 181–204, Zurich, Switzerland, May 26–30, 2024. Springer, Cham, Switzerland.

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

[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.

[FMP23] Tako Boris Fouotsa, Tomoki Moriya, and Christophe Petit. M-SIDH and MD-SIDH: Countering SIDH attacks by masking information. In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part V*, volume 14008 of *LNCS*, pages 282–309, Lyon, France, April 23–27, 2023. Springer, Cham, Switzerland.

[FO99] Eiichiro Fujisaki and Tatsuaki Okamoto. Secure integration of asymmetric and symmetric encryption schemes. In *Proceedings of the 19th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '99, page 537554, Berlin, Heidelberg, 1999. Springer-Verlag.

[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.

[HHK17] Dennis Hofheinz, Kathrin Hövelmanns, and Eike Kiltz. A modular analysis of the Fujisaki-Okamoto transformation. In Yael Kalai and Leonid Reyzin, editors, *TCC 2017, Part I*, volume 10677 of *LNCS*, pages 341–371, Baltimore, MD, USA, November 12–15, 2017. Springer, Cham, Switzerland.

[HLMW26] Arthur Herlédan Le Merdy and Benjamin Wesolowski. Unconditional foundations forăsupersingular isogeny-based cryptography. In Benny Applebaum and Huijia (Rachel) Lin, editors, *Theory of Cryptography*, pages 266–297, Cham, 2026. Springer Nature Switzerland.

[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.

[JZM19] Haodong Jiang, Zhenfeng Zhang, and Zhi Ma. Tighter security proofs for generic key encapsulation mechanism in the quantum random oracle model. In Jintai Ding and Rainer Steinwandt, editors, *Post-Quantum Cryptography - 10th International Conference, PQCrypto 2019*, pages 227–248, Chongqing, China, May 8–10, 2019. Springer, Cham, Switzerland.

[KLPT14] David Kohel, Kristin Lauter, Christophe Petit, and Jean-Pierre Tignol. On the quaternion-isogeny path problem. *LMS Journal of Computation and Mathematics*, 17(A):418–432, 2014.

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

[LLC+24] Kaizhan Lin, Jianming Lin, Shiping Cai, Weize Wang, and Chang-An Zhao. Compressed M-SIDH: an instance of compressed SIDH-like schemes with isogenies of highly composite degrees. *DCC*, 92(6):1823–1843, 2024.

[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.

[MMP+23] Luciano Maino, Chloe Martindale, Lorenz Panny, Giacomo Pope, and Benjamin Wesolowski. A direct key recovery attack on SIDH. In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part V*, volume 14008 of *LNCS*, pages 448–471, Lyon, France, April 23–27, 2023. Springer, Cham, Switzerland.

[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.

[MS25] Tomoki Moriya and Miha Stopar. LIT-SiGamal: An efficient isogeny-based PKE based on a LIT diagram. In *ASIACRYPT 2025, Part IV*, LNCS, pages 275–306. Springer, Singapore, Singapore, December 7–11, 2025.

[NO24] Kohei Nakagawa and Hiroshi Onuki. QFESTA: Efficient algorithms and parameters for FESTA using quaternion algebras. LNCS, pages 75–106, Santa Barbara, CA, USA, August 2024. Springer, Cham, Switzerland.

[Per12] Edoardo Persichetti. *Improving the Efficiency of Code-Based Cryptography*. PhD thesis, University of Auckland, November 2012.

[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.

[PW24] Aurel Page and Benjamin Wesolowski. The supersingular endomorphism ring and one endomorphism problems are equivalent. In Marc Joye and Gregor Leander, editors, *EUROCRYPT 2024, Part VI*, volume 14656 of *LNCS*, pages 388–417, Zurich, Switzerland, May 26–30, 2024. Springer, Cham, Switzerland.

[Rob23] Damien Robert. Breaking SIDH in polynomial time. In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part V*, volume 14008 of *LNCS*, pages 472–503, Lyon, France, April 23–27, 2023. Springer, Cham, Switzerland.

[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.

[SAB+20] Peter Schwabe, Roberto Avanzi, Joppe Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M. Schanck, Gregor Seiler, and Damien Stehlé. CRYSTALS-KYBER. Technical report, National Institute of Standards and Technology, 2020. available at https://csrc.nist.gov/projects/post-quantum-cryptography/post-quantum-cryptography-standardization/round-3-submissions.

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

[Tan09] Seiichiro Tani. Claw finding algorithms using quantum walk. *Theoretical Computer Science*, 410(50):5285–5297, 2009.

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

[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.

CHAPTER A
Overview of Implementation

In this brief chapter, we provide an overview of the different modules required for an implementation of QIMEN-PIKE, 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 and Chapter D.

`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-PIKE.KeyGen, QIMEN-PIKE.Encaps, QIMEN-PIKE.Decaps). We give a full description of this module in Chapter E.

`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 F.

`quat`: provides the core quaternion arithmetic, in particular core algorithms for ideal translation (`qlapoti`). The quaternion routines used by QIMEN-PIKE are described in Sections 2.6.2 and 2.6.3.

`id2iso`: provides the algorithms to translate quaternials to kernels of isogenies, used mainly in QIMEN-PIKE.KeyGen.

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-PIKE.KeyGen, QIMEN-PIKE.Encaps, QIMEN-PIKE.Decaps.

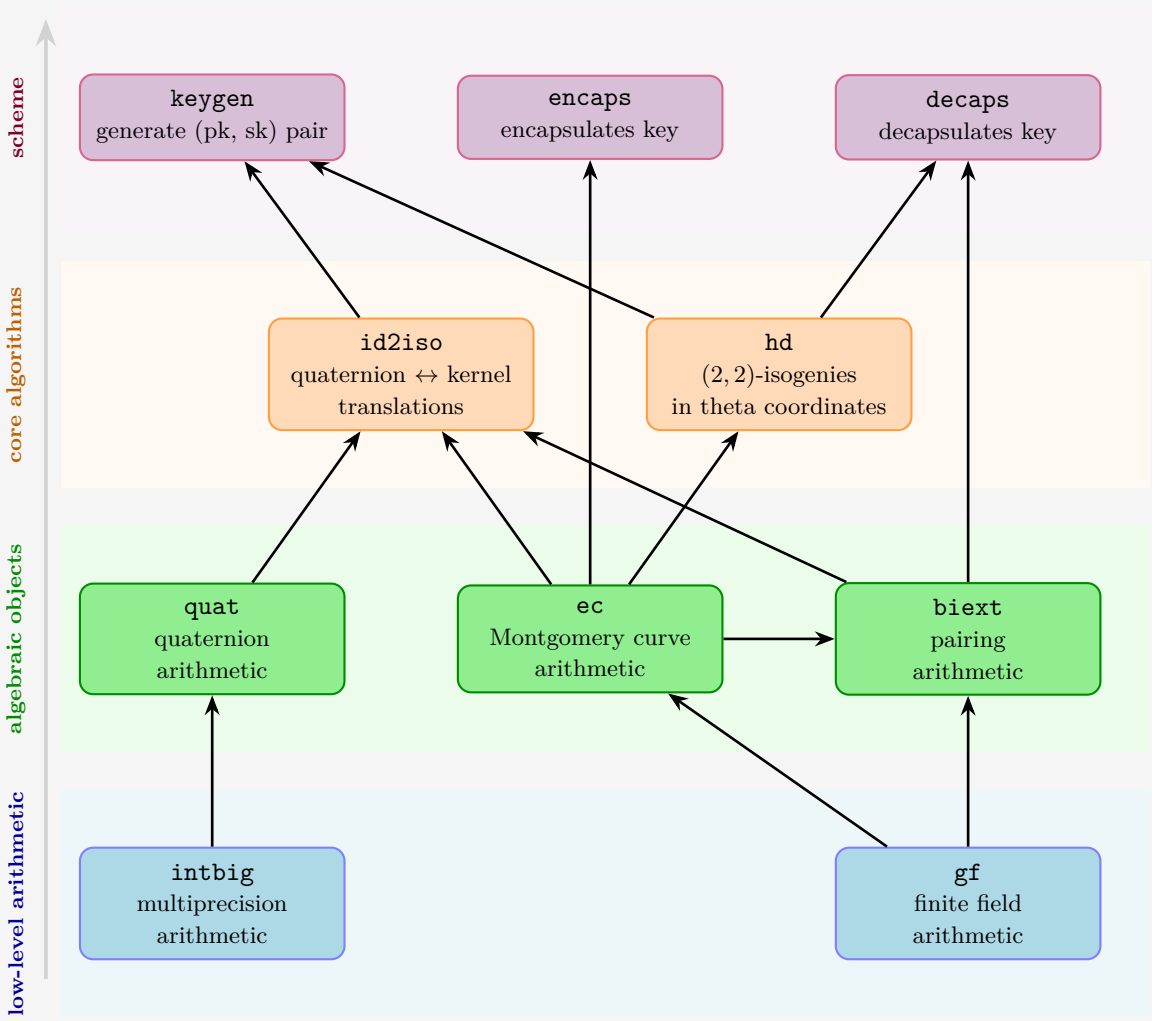

Figure A.1: Module structure of the reference implementation of QIMEN-PIKE, going up in abstraction vertically.

CHAPTER B

Finite field arithmetic (implementation details)*

All higher-level operations in QIMEN-PIKE 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.

The parameters n and r satisfy $n \cdot r \ge \lceil \log_2 p \rceil$; Table B.1 lists the concrete values for each parameter set of Section 5.1.

Table B.1: Field element representation parameters for the reference implementation. Here n is the number of 64-bit limbs and r is the radix (significant bits per limb).

Parameter set	$\lceil \log_2 p \rceil$	Limbs n	Radix r	Useful bits $n{\cdot}r$
NGCC-1	479	8	61	488
NGCC-2	765	13	59	767
NGCC-3	1534	26	60	1560

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 23 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 24 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.2 for use in subsequent sections.

Table B.2: 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 23 Fp2Mul(a, b)

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

1: $t_0 \leftarrow (a_0 + a_1) \cdot (b_0 + b_1)$
2: $t_1 \leftarrow a_1 \cdot b_1$
3: $c_0 \leftarrow a_0 \cdot b_0$
4: $c_1 \leftarrow t_0 - t_1 - c_0$ ▷ $= a_0 b_1 + a_1 b_0$
5: $c_0 \leftarrow c_0 - t_1$ ▷ $= a_0 b_0 - a_1 b_1$
6: **return** $c = c_0 + c_1\, i$

Algorithm 24 Fp2Sqr(a)

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

1: $c_1 \leftarrow 2\, a_0 \cdot a_1$
2: $c_0 \leftarrow (a_0 + a_1) \cdot (a_0 - a_1)$ ▷ $= a_0^2 - a_1^2$
3: **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 $\mathbb{F}_{p^2}^{\times}$. We distinguish the dyadic case from the odd-order case.

B.4.0.1 Subgroups of 2-power order

We first consider subgroups of order 2^e. 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 25 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 $b + 2k' \bmod 2^e$

B.4.0.2 Odd-order subgroups

We now turn to the odd-order case. Let ℓ be an odd prime, let $e \geq 1$, let $\zeta \in \mu_{\ell^e} \subset \mathbb{F}_{p^2}^{\times}$ be an element of order ℓ^e, and let $\eta = \zeta^k$. The goal is to recover $k \bmod \ell^e$. In this case we only need the standard generic methods.

We first consider the prime-order case $e = 1$. Then the problem is to recover $k \in \mathbb{Z}/\ell\mathbb{Z}$ from $\eta = \zeta^k$ in the cyclic group $\langle\zeta\rangle$ of order ℓ. Since the groups arising here are small, a generic algorithm is sufficient. We use baby-step giant-step: precompute the baby steps ζ^j for $0 \leq j < m$, where $m = \lceil\sqrt{\ell}\rceil$, and then search for a giant step of the form $\eta\zeta^{-im}$ that matches one of the stored values. Once a match is found, one obtains $k \equiv im + j \pmod{\ell}$.

Algorithm 26 DLogPrimeOrder(ζ, η, ℓ)

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

$m \leftarrow \lceil\sqrt{\ell}\rceil$
$T \leftarrow \emptyset$
$u \leftarrow 1$
for $j = 0$ to $m - 1$ **do**
  Store (u, j) in T
  $u \leftarrow u \cdot \zeta$
$c \leftarrow \zeta^{-m}$
$v \leftarrow \eta$
for $i = 0$ to $m - 1$ **do**
  if v appears in T with entry (v, j) **then**
    return $im + j \bmod \ell$
  $v \leftarrow v \cdot c$
return failure

For the general prime-power case, write

$$k = k_0 + k_1\ell + \cdots + k_{e-1}\ell^{e-1}, \qquad 0 \leq k_j < \ell.$$

We then use the usual Pohlig–Hellman digit lifting. At stage j, one first removes the contribution of the digits that have already been recovered. One then raises the corrected target to the power ℓ^{e-1-j}, which

places it in the prime-order subgroup generated by $\zeta^{\ell^{e-1}}$. Thus each base-ℓ digit k_j is recovered by one call to Algorithm 26.

Algorithm 27 DLogPrimePower(ζ, η, ℓ, e)

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

Output: $k \in \mathbb{Z}/\ell^e\mathbb{Z}$ such that $\eta = \zeta^k$

$g \leftarrow \zeta^{\ell^{e-1}}$
$k \leftarrow 0$
for $j = 0$ to $e - 1$ **do**
    $u \leftarrow \eta \cdot \zeta^{-k}$
    $v \leftarrow u^{\ell^{e-1-j}}$
    $k_j \leftarrow$ DLogPrimeOrder(g, v, ℓ)
    $k \leftarrow k + k_j \ell^j$
return $k \bmod \ell^e$

Finally, for a general smooth odd order

$$n = \prod_{i=1}^{r} \ell_i^{e_i},$$

one applies Algorithm 27 to each primary component separately and then combines the residues modulo $\ell_i^{e_i}$ with the Chinese remainder theorem.

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 28) 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 29) 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 30) 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 28 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 29 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 30 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 31 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 32 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 33 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 2.2.1.1.
- 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 34 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 35 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 36 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-PIKE 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 38) 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 37) 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 39) 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 37 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 38 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 39 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: 5S + 11M + 7a

C.4 Torsion Basis

For the fixed starting curve $E_0 : y^2 = x^3 + x$, the bases of the torsion subgroups $E_0[2^a]$, $E_0[C]$, and $E_0[D]$ are precomputed and stored as constants. For the dynamically generated curves E_A, E_B, and E_{AB}, the torsion bases are usually obtained by evaluating the precomputed bases of E_0 through the corresponding isogeny chains, rather than regenerating them from scratch.

For 2-power torsion, one can remain entirely in projective x-only coordinates. Starting from the affine Montgomery coefficient A, one searches for a point of suitable shape, forms a second point through the relation $x_{R+S} = -x_R - A$, projects both candidates to $E[2^e]$ by multiplying by the cofactor $(p+1)/2^e$, and then recovers the remaining x-coordinate by ProjectiveDifference. This yields a deterministic basis in the form (x_R, x_S, x_{R+S}), which is well suited to the ladder-based arithmetic used throughout the implementation.

For odd prime-power torsion $E[\ell^e]$, there is generally no comparable x-only shortcut, so the standard approach is formulated in full coordinates. One samples random points, maps them to the ℓ^e-torsion via the cofactor $(p+1)/\ell^e$ [1], and rejects any point with an order strictly less than ℓ^e. A second point is accepted only if it is linearly independent from the first. To handle a general torsion group, the problem is decomposed into the prime-power cases described above, applying a similar strategy to each. Precisely, for a general integer $N = \prod_i \ell_i^{e_i}$, we sample points in $E[N]$ by the same routine. A point $P \in E[N]$ has exact order N if and only if $[N/\ell_i]P \neq 0_E$ for every prime divisor $\ell_i \mid D$. Thus, to obtain a basis of $E[N]$, one samples two points $P, Q \in E[N]$, checks the exact-order condition for both, and finally verifies the linear independence.

Hints. Torsion basis generation can be sped up through the use of hints. In the case of 2-power torsion, generating a basis requires the knowledge of the quadratic residuosity of the curve coefficient A alongside the specific index used to determine x_P. By supplying these values as hints, the protocol streamlines this process: one party employs TorsionBasisToHint to compute both the basis and its associated hints, incorporating the hints directly into the public key or ciphertext. The counterpart then executes TorsionBasisFromHint, which utilizes these hints to efficiently recover 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 E. The hints used for the other cases in TorsionBasisGivenOrderFromHint avoid expensive square root operations and eliminate the need to perform an independence test.

In QIMEN-PIKE.KeyGenCompressed of the compressed QIMEN-PIKE implementation, after recovering the scalars u_{A,C_1} and v_{A,C_1} such that $[2^a]Q_A^+ = [u_{A,C_1}]U_{A,C_1} + [v_{A,C_1}]V_{A,C_1}$, we aim to minimize

[1] For the quadratic twist E_t, replace the cofactor by $(p-1)/\ell^e$.

Algorithm 40 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)

their size to ensure a compact public key. When C_1 is a prime power, this is straightforward: one simply inverts either u_{A,C_1} or v_{A,C_1} and multiplies the other scalar by this inverse. However, in our setting, the torsion order is composite, taking the form $C_1 = \prod_{i=1}^{n} \ell_i^{e_i}$ with $n > 1$. Consequently, it is possible that neither u_{A,C_1} nor v_{A,C_1} is invertible modulo C_1.

To overcome this issue, we follow the approach of [LLC+24, Section 4.3]. The key observation is that at least one of u_{A,C_1} or v_{A,C_1} must be invertible modulo each prime-power factor $\ell_i^{e_i}$ for $i = 1, 2, \ldots, n$. To explicitly track which scalar is inverted for each factor, we introduce an n-bit label $\mathsf{label} = (l_n l_{n-1} \cdots l_1)_2$. As detailed in LabelComputation, we set $l_i = 0$ if u_{A,C_1} is invertible modulo $\ell_i^{e_i}$, and $l_i = 1$ otherwise. By applying the Chinese remainder theorem, the algorithm aggregates these local inversions to output the label label alongside global scalars $m, s \in \mathbb{Z}/C_1\mathbb{Z}$ satisfying

$$m = \begin{cases} u_{A,C_1}^{-1} \bmod \ell_i^{e_i}, & \text{if } l_i = 0, \\ v_{A,C_1}^{-1} \bmod \ell_i^{e_i}, & \text{otherwise,} \end{cases}, s = \begin{cases} m v_{A,C_1} \bmod \ell_i^{e_i}, & \text{if } l_i = 0, \\ m u_{A,C_1} \bmod \ell_i^{e_i}, & \text{otherwise,} \end{cases}, \tag{C.1}$$

and the corresponding label label.

Using only the compressed scalar s and the label label, ScalarRecoveryFromLabel reconstructs two full scalars $s_1, s_2 \in \mathbb{Z}/C_1\mathbb{Z}$ such that

$$s_1 = \begin{cases} m v_{A,C_1} \bmod \ell_i^{e_i}, & \text{if } l_i = 0, \\ 1 \bmod \ell_i^{e_i}, & \text{otherwise,} \end{cases}, s_2 = \begin{cases} 1 \bmod \ell_i^{e_i}, & \text{if } l_i = 0, \\ m u_{A,C_1} \bmod \ell_i^{e_i}, & \text{otherwise.} \end{cases} \tag{C.2}$$

Following the proof of [LLC+24, Proposition 4], it is straightforward to verify that the point $[C_1/\ell_i^{e_i}]([s_1]U_{A,C_1} + [s_2]V_{A,C_1})$ generates the subgroup $\langle [C_1/\ell_i^{e_i}]([2^a]Q_A^+) \rangle$ for $i = 1, 2, \cdots, n$. This property confirms the correctness of compressed encryption without requiring explicit knowledge of the coordinates of Q_A^+.

Algorithm 41 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})

Algorithm 42 TorsionBasisGivenOrderToHint(E, N, b)

Input: Supersingular Montgomery curve $E/\mathbb{F}_{p^2}$ in projective Montgomery form $(A : C)$, an integer $N = \prod_{i=1}^{n} \ell_i^{e_i}$ and a flag flag indicating whether the basis is defined on the curve $E(\mathbb{F}_{p^2})$ or its quadratic twist
Output: An x-only basis $(P, Q, P - Q)$ of $E[N]$, together with hint $h = (h_0, h_1)$

$h_0 \leftarrow 0$, found $\leftarrow$ false
Let the cofactor c_N be an integer used to project points to the N-torsion
while found = false **do**
  $h_0 \leftarrow h_0 + 1$
  $P \leftarrow (1 + h_0 i : 1)$
  $F \leftarrow X(P)(C^2 X(P)^2 + ACX(P) + C^2)$
  if lssquare$(F) \neq$ flag **then**
    continue
  $P \leftarrow [c_N]P$
  $P' \leftarrow [\prod_{i=1}^{n} \ell_i^{e_i - 1}]P$
  if P' has nonzero projection on every ℓ_i-component **then**
    found $\leftarrow$ true
$h_1 \leftarrow 0$, found $\leftarrow$ false
while found = false **do**
  $h_1 \leftarrow h_1 + 1$
  $Q \leftarrow (1 + (h_0 + h_1)i : 1)$
  $F \leftarrow X(Q)(C^2 X(Q)^2 + ACX(Q) + C^2)$
  if lssquare$(F) \neq$ flag **then**
    continue
  $Q \leftarrow [c_N]Q$
  $Q' \leftarrow [\prod_{i=1}^{n} \ell_i^{e_i - 1}]Q$
  if Q' is independent from P' **then**
    found $\leftarrow$ true
$T \leftarrow$ ProjectiveDifference(P, Q, E)
return $(P, Q, T), (h_0, h_1)$

Algorithm 43 TorsionBasisGivenOrderFromHint$(E, N, hint)$

Input: Supersingular Montgomery curve $E/\mathbb{F}_{p^2}$, an integer N and hint $h = (h_0, h_1)$
Output: An x-only basis $\mathcal{B} = (P, Q, P - Q)$ of $E[N]$

Let the cofactor c_N be an integer used to project points to the N-torsion
$P \leftarrow (1 + h_0 i : 1)$
$P \leftarrow [c_N]P$
$Q \leftarrow (1 + (h_0 + h_1)i : 1)$
$Q \leftarrow [c_N]Q$
$T \leftarrow$ ProjectiveDifference(P, Q, E)
return (P, Q, T)

Algorithm 44 LabelComputation(N, u, v)

Input: An integer $N = \prod_{i=1}^{n} \ell_i^{e_i}$ and two scalars $u, v \in \mathbb{Z}/N\mathbb{Z}$
Output: Scalars $m, s \in \mathbb{Z}/N\mathbb{Z}$ and an n-bit label $\mathsf{label} = (l_n l_{n-1} \cdots l_1)_2$ such that

$$m = \begin{cases} u^{-1} \bmod \ell_i^{e_i}, \text{ if } l_i = 0, \\ v^{-1} \bmod \ell_i^{e_i}, \text{otherwise,} \end{cases}, s = \begin{cases} mv \bmod \ell_i^{e_i}, \text{ if } l_i = 0, \\ mu \bmod \ell_i^{e_i}, \text{otherwise,} \end{cases}$$

$label \leftarrow 0$
for $i = 1, 2, \cdots, n$ **do**
    if $u \bmod \ell_i = 0$ **then**
        $label = label + 2^i$, $m' \leftarrow v^{-1} \bmod \ell^{e_i}$, $s' \leftarrow m' \cdot u$
    else
        $m' \leftarrow u^{-1} \bmod \ell^{e_i}$, $s' \leftarrow m' \cdot v$
    if $i = 1$ **then**
        $m \leftarrow m'$, $s \leftarrow s'$, $m'' \leftarrow m'$, $s'' \leftarrow s'$, $M \leftarrow \ell_1^{e_1}$
    else
        Compute m such that $m \equiv m' \bmod \ell_i^{e_i}$ and $m \equiv m'' \bmod M$
        Compute s such that $s \equiv s' \bmod \ell_i^{e_i}$ and $s \equiv s'' \bmod M$
        $m'' \leftarrow m, s'' \leftarrow s, M \leftarrow M \cdot \ell_i^{e_i}$
return m, s, label

Algorithm 45 ScalarRecoveryFromLabel$(N, s, label)$

Input: An integer $N = \prod_{i=1}^{n} \ell_i^{e_i}$, a scalar $s \in \mathbb{Z}/N\mathbb{Z}$ and an n-bit label $\mathsf{label} = (l_n l_{n-1} \cdots l_1)_2$
Output: Scalars $s_1, s_2 \in \mathbb{Z}/N\mathbb{Z}$ such that

$$s_1 = \begin{cases} s \bmod \ell_i^{e_i}, \text{ if } l_i = 0, \\ 1 \bmod \ell_i^{e_i}, \text{otherwise,} \end{cases}, s_2 = \begin{cases} 1 \bmod \ell_i^{e_i}, \text{ if } l_i = 0, \\ s \bmod \ell_i^{e_i}, \text{otherwise,} \end{cases}$$

for $i = 1, 2, \cdots, n$ **do**
  if $l_i = 1$ **then**
    $s_1' \leftarrow 1$, $s_2' \leftarrow s \bmod \ell^{e_i}$
  else
    $s_1' \leftarrow s \bmod \ell^{e_i}$, $s_2' \leftarrow 1$
  if $i = 1$ **then**
    $s_1 \leftarrow s_1'$, $s_2 \leftarrow s_2'$, $s_1'' \leftarrow s_1'$, $s_2'' \leftarrow s_2'$, $M \leftarrow \ell_1^{e_1}$
  else
    Compute s_1 such that $s_1 \equiv s_1' \bmod \ell_i^{e_i}$ and $s_1 \equiv s_1'' \bmod M$
    Compute s_2 such that $s_2 \equiv s_2' \bmod \ell_i^{e_i}$ and $s_2 \equiv s_2'' \bmod M$
    $s_1'' \leftarrow s_1, s_2'' \leftarrow s_2, M \leftarrow M \cdot \ell_i^{e_i}$
return s_1, s_2

CHAPTER D

1-Dimensional isogenies (implementation details)*

QIMEN-PIKE uses smooth-degree isogenies in dimension 1, decomposed into chains of small prime-degree steps. Besides, QIMEN-PIKE also requires smooth odd-degree isogenies for evaluating endomorphisms. The available odd degrees factor into small prime powers dividing $p+1$ or $p-1$. For each prime $\ell = 2d+1$, the isogeny computation uses a hybrid Montgomery–Twisted Edwards formula following Costello and Hisil [CH17]. Montgomery projective coordinates $(X : Z)$ are mapped to Twisted Edwards coordinates via $y = (X - Z, X + Z)$. Differential doubling and addition in the Edwards model (Algorithms 46 and 47) are used to build the kernel point set.

Algorithm 46 $\textsc{yDBL}(P, (A : C))$

Require: Twisted Edwards point $P = (Y_P : Z_P)$, Montgomery constants $(A : C)$ of curve E **Ensure:** $[2]P = (Y_{2P} : Z_{2P})$ in Twisted Edwards form

1. $t_0 \leftarrow Y_P^2$
2. $t_1 \leftarrow Z_P^2$
3. $Z \leftarrow C \cdot t_0$
4. $X \leftarrow Z \cdot t_1$
5. $t_1 \leftarrow t_1 - t_0$
6. $t_0 \leftarrow A \cdot t_1$
7. $Z \leftarrow Z + t_0$
8. $Z \leftarrow Z \cdot t_1$
9. $Y_{2P} \leftarrow X - Z$
10. $Z_{2P} \leftarrow X + Z$
11. **return** $(Y_{2P} : Z_{2P})$

▷ Cost: 2S + 3M + 3a

Algorithm 47 $\textsc{yADD}(P, Q, P-Q)$

Require: Twisted Edwards points $P = (Y_P : Z_P)$, $Q = (Y_Q : Z_Q)$, $P - Q = (Y_{PQ} : Z_{PQ})$
Ensure: $P + Q = (Y_R : Z_R)$ in Twisted Edwards form

1. $a \leftarrow Z_P \cdot Y_Q$
2. $b \leftarrow Y_P \cdot Z_Q$
3. $c \leftarrow (a + b)^2$
4. $d \leftarrow (a - b)^2$
5. $X \leftarrow (Z_{PQ} - Y_{PQ}) \cdot c$
6. $Z \leftarrow (Z_{PQ} + Y_{PQ}) \cdot d$
7. $Y_R \leftarrow X - Z$
8. $Z_R \leftarrow X + Z$
9. **return** $(Y_R : Z_R)$

▷ Cost: 2S + 2M + 6a

The kernel point set is then constructed by Algorithm 48.

Algorithm 48 $\textsc{KernelPointSet}(\ell, P, (A : C))$

Input: Odd prime $\ell = 2d + 1$; kernel generator $P = (X_P : Z_P)$ in Montgomery form; curve constants $(A : C)$
Output: Kernel point set $\{K_j\}_{j=0}^{d-1}$ in Twisted Edwards form

$K_0 \leftarrow (X_P - Z_P,\ X_P + Z_P)$ ▷ Montgomery → Edwards
$K_1 \leftarrow \textsc{yDBL}(K_0, (A : C))$
for $j = 2, \ldots, d-1$ **do**
    $K_j \leftarrow \textsc{yADD}(K_{j-1}, K_0, K_{j-2})$ ▷ $K_j = [j+1]P$
return $\{K_0, K_1, \ldots, K_{d-1}\}$

OddIsogenyCodomain (Algorithm 49) computes the codomain curve by forming the product of all kernel points in the Edwards model, raising the Edwards curve parameters a and d to the ℓ-th power by left-to-right binary exponentiation, and mapping back to Montgomery form.

Algorithm 49 OddIsogenyCodomain$(\ell, (A : C))$

Input: Odd prime $\ell = 2d + 1$; Montgomery constants $(A : C)$; kernel set $\{K_j\}_{j=0}^{d-1}$ (precomputed)
Output: Codomain Montgomery coefficient $(A' : C')$

$B_y \leftarrow K_0.y; \quad B_z \leftarrow K_0.z$ ▷ accumulate product
for $j = 1, \ldots, d-1$ **do**
  $B_y \leftarrow B_y \cdot K_j.y; \quad B_z \leftarrow B_z \cdot K_j.z$
$d_E \leftarrow A - C$ ▷ Edwards parameter d
$a_E \leftarrow A; \quad d'_E \leftarrow d_E$
for $j = 1, \ldots, \lceil \log_2 \ell \rceil - 1$ **do** ▷ binary exp. to compute a_E^ℓ, d_E^ℓ
  $a_E \leftarrow a_E^2; \quad d'_E \leftarrow d'^2_E$
  if bit j of ℓ is 1 **then**
    $a_E \leftarrow a_E \cdot A; \quad d'_E \leftarrow d'_E \cdot d_E$
$B_y \leftarrow B_y^8; \quad B_z \leftarrow B_z^8$ ▷ raise to 8th power (three squarings)
$A' \leftarrow a_E \cdot B_z$
$C' \leftarrow A' - d'_E \cdot B_y$
return $(A' : C')$

OddIsogenyEval (Algorithm 51) evaluates a point through the isogeny using the CrissCross subroutine of Costello and Hisil.

Algorithm 50 CrissCross$(\alpha, \beta, \gamma, \delta)$

Input: $\alpha, \beta, \gamma, \delta \in \mathbb{F}_{p^2}$
Output: (r_0, r_1) with $r_0 = \alpha\delta + \beta\gamma$, $r_1 = \alpha\delta - \beta\gamma$

$t_1 \leftarrow \alpha \cdot \delta; \quad t_2 \leftarrow \beta \cdot \gamma$
$r_0 \leftarrow t_1 + t_2; \quad r_1 \leftarrow t_1 - t_2$
return (r_0, r_1) ▷ Cost: 2M + 2a

Algorithm 51 OddIsogenyEval(ℓ, P)

Input: Odd prime $\ell = 2d + 1$; projective point $P = (X_P : Z_P)$; kernel set $\{K_j\}_{j=0}^{d-1}$ (precomputed)
Output: Image $Q = (X_Q : Z_Q) \in E'$ of P under the ℓ-isogeny

$S_0 \leftarrow X_P + Z_P; \quad S_1 \leftarrow X_P - Z_P$
$(R_0, R_1) \leftarrow$ CrissCross$(K_0.z,\ K_0.y,\ S_0,\ S_1)$
for $j = 1, \ldots, d-1$ **do**
  $(T_0, T_1) \leftarrow$ CrissCross$(K_j.z,\ K_j.y,\ S_0,\ S_1)$
  $R_0 \leftarrow R_0 \cdot T_0; \quad R_1 \leftarrow R_1 \cdot T_1$
$R_0 \leftarrow R_0^2; \quad R_1 \leftarrow R_1^2$
$X_Q \leftarrow X_P \cdot R_0; \quad Z_Q \leftarrow Z_P \cdot R_1$
return $(X_Q : Z_Q)$

In QIMEN-PIKE, an odd-degree isogeny chain arising in the protocol is decomposed into subchains of degree ℓ^e, where ℓ is an odd prime. We therefore describe the computation in two subroutines: a top-level routine for the full odd chain, and a recursive routine for each odd prime-power subchain. Each prime-power subchain is then evaluated using a balanced strategy.

Algorithm 52 OddIsogenyChain(E, K, N, pts)

Input: A Montgomery curve E, a generator K of order $N = \prod_{i=1}^{n} \ell_i^{e_i}$ the odd-kernel components on E, and a list of points $\text{pts} = (Q_1, \ldots, Q_m)$ to be evaluated.

Output: The image curve E' and the images of all points in pts under the odd isogeny.

eval_pts $\leftarrow (Q_1, \ldots, Q_m, K)$
for $i = 1$ to n **do**
    $P \leftarrow$ eval_pts$[m+1]$
    for $j = i+1$ to n **do**
        $P \leftarrow$ Ladder$(P, E, \ell_j^{e_j})$
    $\mathcal{S} \leftarrow ()$
    PrimePowerRec$(E, P, i, e_i, \mathcal{S}, \text{eval_pts})$
pts $\leftarrow$ (eval_pts$[1], \ldots,$ eval_pts$[m]$)
return (E, pts)

Algorithm 53 PrimePowerRec$(E, P, \ell_i, h, \mathcal{S}, \text{pts})$

Input: Montgomery curve $E = (A : C)$, a point P of order ℓ_i^h, a stack $\mathcal{S}$ of kernel points, and a list pts of points need to evaluated.

Output: E is replaced by the image curve after the corresponding ℓ_i^h-subchain, and every point in $\mathcal{S} \cup \text{pts}$ is replaced by its image.

if $h = 0$ **then**
    return
if $h = 1$ **then**
    $\mathcal{K} \leftarrow$ KernelPointSet$(\ell_i, P, (A : C))$
    $E_{\text{new}} \leftarrow$ OddIsogenyCodomain$(\ell_i, (A : C))$
    for each $S \in \mathcal{S}$ **do**
        $S \leftarrow$ OddIsogenyEval(ℓ_i, S)
    for each $Q \in$ pts **do**
        $Q \leftarrow$ OddIsogenyEval(ℓ_i, Q)
    $E \leftarrow E_{\text{new}}$
    return
$r \leftarrow \lfloor h/1.5 \rfloor, \quad s \leftarrow h - r$
push P onto $\mathcal{S}$
$P_{\text{right}} \leftarrow$ Ladder(P, E, ℓ_i^s)
PrimePowerRec$(E, P_{\text{right}}, i, r, \mathcal{S}, \text{pts})$
$P_{\text{left}} \leftarrow \text{top}(\mathcal{S})$, and pop it from $\mathcal{S}$
PrimePowerRec$(E, P_{\text{left}}, i, s, \mathcal{S}, \text{pts})$

CHAPTER E

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 E.1 introduces theta coordinates of level 2, and
- Section E.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 E.4,
2. We iterate generic $(2, 2)$-isogenies entirely in theta coordinates, see Section E.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 E.5.

E.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.

E.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 54.

Algorithm 54 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

E.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.

E.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.

E.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 55 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 56 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

E.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 E.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 E.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 E.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 E.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.

E.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 57 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 58 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

1. $t_1 \leftarrow x_P \cdot \alpha^{-1}$	2. $t_2 \leftarrow y_P \cdot \beta^{-1}$	3. **if** $t_1 \neq t_2$, **return false**
4. $t_1 \leftarrow z_P \cdot \gamma^{-1}$	5. $t_2 \leftarrow w_P \cdot \delta^{-1}$	6. **if** $t_1 \neq t_2$, **return false**
7. $t_1 \leftarrow x_Q \cdot \alpha^{-1}$	8. $t_2 \leftarrow z_Q \cdot \gamma^{-1}$	9. **if** $t_1 \neq t_2$, **return false**
10. $t_1 \leftarrow y_Q \cdot \beta^{-1}$	11. $t_2 \leftarrow w_Q \cdot \delta^{-1}$	12. **if** $t_1 \neq t_2$, **return false**

13. **return true** ▷ Cost: at most 8M

E.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 59 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

1. $(x\alpha\beta, \_, x\gamma\delta, \_) \leftarrow \mathcal{H} \circ \mathcal{S}(x_{T_1'}, y_{T_1'}, z_{T_1'}, w_{T_1'})$
2. $(\alpha^2, \beta^2, \gamma^2, \delta^2) \leftarrow \mathcal{H} \circ \mathcal{S}(a, b, c, d)$
3. $\alpha\beta \leftarrow \sqrt{\alpha^2\beta^2}$
4. $\alpha\gamma \leftarrow \sqrt{\alpha^2\gamma^2}$
5. $\beta \leftarrow \alpha\beta \cdot \alpha\gamma$
6. $\delta^{-1} \leftarrow \beta \cdot x\gamma\delta$
7. $\beta \leftarrow \beta \cdot x\alpha\beta$
8. $\delta \leftarrow x\gamma\delta \cdot \alpha\beta \cdot \alpha^2$
9. $\alpha \leftarrow x\alpha\beta \cdot \alpha^2$
10. $\gamma \leftarrow \alpha \cdot \gamma^2$
11. $\alpha \leftarrow \alpha \cdot \alpha\gamma$
12. $\alpha^{-1} \leftarrow x\alpha\beta \cdot \delta^2$
13. $\gamma^{-1} \leftarrow \alpha^{-1} \cdot \beta^2$
14. $\alpha^{-1} \leftarrow \alpha^{-1} \cdot \gamma^2$
15. $\beta^{-1} \leftarrow \alpha^{-1} \cdot \alpha\beta$
16. $\alpha^{-1} \leftarrow \alpha^{-1} \cdot \beta^2$
17. $\gamma^{-1} \leftarrow \gamma^{-1} \cdot \alpha\gamma$
18. $\delta^{-1} \leftarrow \delta^{-1} \cdot \beta^2$
19. $(a_2, b_2, c_2, d_2) \leftarrow \mathcal{H}(\alpha, \beta, \gamma, \delta)$
20. $0_B \leftarrow (a_2 : b_2 : c_2 : d_2)$
21. **return** $(\alpha : \beta : \gamma : \delta),\ (\alpha^{-1} : \beta^{-1} : \gamma^{-1} : \delta^{-1}),\ 0_B$

E.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 60 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

1. $(\alpha^2, \beta^2, \gamma^2, \delta^2) \leftarrow \mathcal{H} \circ \mathcal{S}(a, b, c, d)$
2. $\alpha \leftarrow \alpha^2$
3. $\beta \leftarrow \alpha^2 \cdot \beta^2$
4. $\gamma \leftarrow \alpha^2 \cdot \gamma^2$
5. $\delta \leftarrow \alpha^2 \cdot \delta^2$
6. $\beta \leftarrow \sqrt{\beta}$
7. $\gamma \leftarrow \sqrt{\gamma}$
8. $\delta \leftarrow \sqrt{\delta}$
9. $\alpha^{-1} \leftarrow \gamma^2 \cdot \delta^2$
10. $\beta^{-1} \leftarrow \alpha^{-1} \cdot \beta$
11. $\alpha^{-1} \leftarrow \alpha^{-1} \cdot \beta^2$
12. $\gamma^{-1} \leftarrow \delta^2 \cdot \beta^2 \cdot \gamma$
13. $\delta^{-1} \leftarrow \gamma^2 \cdot \beta^2 \cdot \delta$
14. $(a_2, b_2, c_2, d_2) \leftarrow \mathcal{H}(\alpha, \beta, \gamma, \delta)$
15. $0_B \leftarrow (a_2 : b_2 : c_2 : d_2)$
16. **return** $(\alpha : \beta : \gamma : \delta),\ (\alpha^{-1} : \beta^{-1} : \gamma^{-1} : \delta^{-1}),\ 0_B$

E.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 61 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 62 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

E.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.

E.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 63 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 64 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 65 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 $(\alpha : \beta : \gamma : 0),\ (\alpha^{-1} : \beta^{-1} : \gamma^{-1} : 0),\ 0_A,\ J,\ N$

E.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-PIKE.

Algorithm 66 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 67 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 $(x_{\Phi(P)} : y_{\Phi(P)} : z_{\Phi(P)} : w_{\Phi(P)})$

E.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 68 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 69 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 70 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 71 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** $(X_1 : Z_1), (X_2 : Z_2)$

E.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 72 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 73 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 74 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 75 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

1: $M \leftarrow$ SplittingIsomorphism(0_A)
2: $0_A \leftarrow M \cdot 0_A$
3: pts $\leftarrow [\, M \cdot R \mid R \in \text{pts}\,]$
4: $(A_3 : C_3), (A_4 : C_4) \leftarrow$ ThetaToProduct(0_A)
5: pts $\leftarrow [\,$ ThetaProductPointToMontgomery$(R, 0_A) \mid R \in \text{pts}\,]$
6: Let E_3, E_4 be the elliptic curves defined by Montgomery coefficients $(A_3 : C_3)$ and $(A_4 : C_4)$
7: **return** $E_3 \times E_4$, 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 76 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 $[\Phi(R) \mid R \in \text{pts}]$

1: $(0_A, \text{consts}, \text{pts}, \text{strat\_pts}, \text{orders}) \leftarrow$ GluingStep(P, Q, e, pts)
2: $(0_A, \text{pts}) \leftarrow$ GenericStep$(0_A, \text{consts}, \text{pts}, \text{strat_pts}, \text{orders})$
3: $(E_3 \times E_4, \text{pts}) \leftarrow$ SplittingStep$(0_A, \text{pts})$
4: **return** $E_3 \times E_4$, pts

CHAPTER F

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.

F.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 77 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 78 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 79 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 80 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** $X(P_2)/X(P_1)$

F.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 81 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 82 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 $\lambda^{(p^2-1)/2^e}$

Algorithm 83 Dlog2Single$(E, e, x(P), x(Q), x(R), x(P-Q), x(P-R), x(R-Q))$

Input: Supersingular Montgomery curve $E/\mathbb{F}_{p^2}$; integer e; x-coordinates of $P, Q, R \in E[2^e]$; $x(P-Q)$, $x(P-R)$, $x(R-Q)$
Output: Scalars $a, b \in \mathbb{Z}/2^e\mathbb{Z}$ such that $R = [a]P + [b]Q$

$S_P \leftarrow (x(P) : 1)$, $S_Q \leftarrow (x(Q) : 1)$, $S_R \leftarrow (x(R) : 1)$
$(U_1, V_1, D_1) \leftarrow (P, Q, x(P-Q))$
$(U_2, V_2, D_2) \leftarrow (P, R, x(P-R))$
$(U_3, V_3, D_3) \leftarrow (R, Q, x(R-Q))$
for $j \leftarrow 1$ **to** 3 **do**
    $L_j \leftarrow (D_j, 1)$
    $M_j \leftarrow (D_j, 1)$
for $i \leftarrow 1$ **to** $e-1$ **do**
    for $j \leftarrow 1$ **to** 3 **do**
        $L_j \leftarrow$ CubicalDiffAdd$(E, L_j, S_{U_j}, x(V_j))$
        $M_j \leftarrow$ CubicalDiffAdd$(E, M_j, S_{V_j}, x(U_j))$
    $S_P \leftarrow$ CubicalDbl(E, S_P)
    $S_Q \leftarrow$ CubicalDbl(E, S_Q)
    $S_R \leftarrow$ CubicalDbl(E, S_R)
$S'_P \leftarrow$ CubicalTranslate(S_P, S_P)
$S'_Q \leftarrow$ CubicalTranslate(S_Q, S_Q)
$S'_R \leftarrow$ CubicalTranslate(S_R, S_R)
for $j \leftarrow 1$ **to** 3 **do**
    $L_j \leftarrow$ CubicalTranslate(L_j, S_{U_j})
    $M_j \leftarrow$ CubicalTranslate(M_j, S_{V_j})
    $\theta_j \leftarrow$ CubicalRatio(M_j, S'_{V_j})/CubicalRatio(L_j, S'_{U_j})
$\eta_b \leftarrow \theta_2$
$\eta_a \leftarrow \theta_3$
return (ω, η_b, η_a)

F.3 Odd-degree pairings

Let $\ell \geq 3$ be odd, with $\ell \nmid p$, and let $P \in E[\ell]$, $Q \in E$. In this case, cubical arithmetic directly gives the *square* of the non-reduced Tate pairing. More precisely, a cubical ladder of length ℓ produces affine lifts of $[\ell]P$ and $[\ell]P + Q$, and the corresponding monodromy ratio yields $e_{T,\ell}(P, Q)^2$ up to ℓ-th powers.

Since 2 is invertible modulo ℓ, this does not cause any difficulty for the reduced Tate pairing. After the final exponentiation, one recovers the pairing itself by an additional exponentiation by $2^{-1} \bmod \ell$. In the same way, the square of the Weil pairing is obtained as the quotient of two Tate pairings.

Algorithm 84 CubicalLadder$(E, n, x(P), x(Q), x(P-Q))$

Input: Montgomery curve $E : y^2 = x^3 + Ax^2 + x$; integer $n = \sum_{i=0}^{b-1} n_i 2^i$ with $n_b = 0$; normalized inputs $(x(P) : 1)$, $(x(Q) : 1)$, $(x(P-Q) : 1)$

Output: Cubical points $[\widetilde{n]}P$ and $\widetilde{[n]P + Q}$

$S_0 \leftarrow (1, 0)$
$S_1 \leftarrow (x(P) : 1)$
$T \leftarrow (x(Q) : 1)$
$d_0 \leftarrow x(Q)$
$d_1 \leftarrow x(P - Q)$
for $i \leftarrow b - 1$ **downto** 0 **do**
    $R \leftarrow$ CubicalDiffAdd$(E, S_0, S_1, x(P))$
    if $n_i \oplus n_{i+1} = 1$ **then**
        swap S_0, S_1
        swap d_0, d_1
    $T \leftarrow$ CubicalDiffAdd(E, T, S_0, d_0)
    $S_0 \leftarrow$ CubicalDbl(E, S_0)
    $S_1 \leftarrow R$
return (S_0, T)

Algorithm 85 TateOdd$(E, N, x(P), x(Q), x(P-Q))$

Input: Supersingular Montgomery curve $E/\mathbb{F}_{p^2}$; odd integer N with $N \nmid p$; x-coordinates of $P, Q, P - Q$ with $P \in E[N]$

Output: Reduced Tate pairing $t_N(P, Q) \in \mu_N$

$(\widetilde{O}, \widetilde{Q'}) \leftarrow$ CubicalLadder$(E, N, x(P), x(Q), x(P-Q))$
$s \leftarrow$ CubicalRatio$((x(Q) : 1), \widetilde{Q'})$ / CubicalRatio$((1 : 0), \widetilde{O})$
$\lambda \leftarrow s^{(p^2-1)/\ell}$
return λ

Algorithm 86 DlogOddSingle$(E, C, x(P), x(Q), x(P-Q), R)$

Input: Supersingular Montgomery curve $E/\mathbb{F}_{p^2}$; $x-$coordinates of points $P, Q, P - Q, R \in E[N]$

Output: Scalars $(a, b) \in (\mathbb{Z}/N\mathbb{Z})^2$ such that $R = [a]P + [b]Q$

$\omega \leftarrow$ TateOdd$(E, N, x(P), x(Q), x(P-Q))$
$\eta_b \leftarrow$ TateOdd$(E, N, x(P), x(R), x(P-R))$
$\eta_a \leftarrow$ TateOdd$(E, N, x(R), x(Q), x(R-Q))$
$a \leftarrow \log_\omega \eta_a$
$b \leftarrow \log_\omega \eta_b$
if $[a]P + [b]Q \neq R$ **then**
    $a \leftarrow N - a$
return (a, b)

Algorithm 87 DlogOdd$(E, N, x(P), x(Q), x(P-Q), x(P_1), x(P_2))$

Input: Supersingular Montgomery curve $E/\mathbb{F}_{p^2}$; $x-$coordinates of points $P, Q, P-Q, P_1, P_2 \in E[N]$
Output: Scalars $(a_1, b_1, a_2, b_2) \in (\mathbb{Z}/N\mathbb{Z})^4$ such that $P_1 = [a_1]P + [b_1]Q$ and $P_2 = [a_2]P + [b_2]Q$

$\omega \leftarrow$ TateOdd$(E, N, x(P), x(Q), x(P-Q))$
$\eta_{b_1} \leftarrow$ TateOdd$(E, N, x(P), x(P_1), x(P-P_1))$
$\eta_{a_1} \leftarrow$ TateOdd$(E, N, x(P_1), x(Q), x(P_1-Q))$
$\eta_{b_2} \leftarrow$ TateOdd$(E, N, x(P), x(P_2), x(P-P_2))$
$\eta_{a_2} \leftarrow$ TateOdd$(E, N, x(P_2), x(Q), x(P_2-Q))$
$a_1 \leftarrow \log_\omega \eta_{a_1}$
$a_2 \leftarrow \log_\omega \eta_{a_2}$
$b_1 \leftarrow \log_\omega \eta_{b_1}$
$b_2 \leftarrow \log_\omega \eta_{b_2}$
if $[a_1]P + [b_1]Q \neq P_1$ **then**
    $a_1 \leftarrow N - a_1$
if $[a_2]P + [b_2]Q \neq P_2$ **then**
    $a_2 \leftarrow N - a_2$
return (a_1, b_1, a_2, b_2)