Exposing SIMD Parallelism in SQIsign: An AVX-512 Implementation

Weize Wang, Chutong Wang, Yu Wu, Qifan Xue, Jieyu Zheng, and Yunlei Zhao

Fudan University, Shanghai, China,
{wzwang23,ctwang24,wuyu25,qfxue24,jyzheng23}@m.fudan.edu.cn,ylzhao@fudan.edu.cn

Abstract. Modern isogeny-based cryptosystems spend much of their running time in finite-field, elliptic-curve, and higher-dimensional isogeny arithmetic. Exploiting SIMD parallelism in these computations is nevertheless nontrivial: central routines such as Montgomery ladders contain loop-carried dependencies, while point, pairing, and theta-coordinate formulas expose only irregular fine-grained parallelism. We show that substantial SIMD parallelism can be recovered by reorganizing the arithmetic dependency graphs of these higher-level primitives rather than vectorizing field multiplication in isolation.

We develop an end-to-end AVX-512IFMA implementation of SQIsign in which data remain in a radix-2^{51} vector representation across most of the curve-side computation. Our redesign includes projective xDBLADD schedules for Montgomery ladders, batched point doubling in several coordinate systems, a vectorized biscalar ladder, fused cubical-arithmetic pairing steps, and batched one- and two-dimensional isogeny evaluation. Relative to the reference C implementation, our implementation achieves end-to-end speedups of 1.76×, 1.71×, and 3.18× for key generation, signing, and verification, respectively, at NIST security level I; combining the same implementation with Qlapoti increases the key-generation and signing speedups to 2.90× and 2.69×.

To test whether these techniques are specific to SQIsign, we further apply the same AVX-512IFMA backend and higher-dimensional vectorization methodology to CORAL, a recent isogeny group action for post-quantum non-interactive key exchange based on two-dimensional 2-isogenies. Across the five parameter sets in our experiments, this yields 1.28–1.40× speedups for key generation and 1.92–2.46× speedups for shared-key computation over the reference C implementation. These results provide cross-scheme evidence that algorithm-level SIMD scheduling is a reusable optimization dimension for higher-dimensional isogeny cryptography.

Keywords: SQIsign · CORAL · SIMD Vectorization · Isogenies · AVX-512

1 Introduction

Isogeny-based cryptography offers some of the most compact public keys and signatures among post-quantum proposals, but this compactness comes with

substantial computational cost. SQIsign [AAA+24], the isogeny-based signature scheme in the NIST additional digital-signature process, is a prominent example. Its Fiat–Shamir signature protocol ultimately translates quaternion ideals into explicit isogenies, and therefore repeatedly invokes arithmetic over $\mathbb{F}_p$ and $\mathbb{F}_{p^2}$, elliptic-curve scalar multiplication, pairings, and isogenies between principally polarized abelian surfaces (PPAS). In current implementations, the *IdealToIsogeny* procedure is a major source of signing cost [BDD+24].

A natural way to accelerate these computations is SIMD vectorization. At the finite-field level, wide-vector architectures can evaluate several independent multiplications in parallel. The main obstacle is that SQIsign is not simply a collection of independent field operations. Montgomery ladders have strict loop-carried dependencies; a single point doubling exposes only limited instruction-level parallelism; pairing computations repeatedly reuse state across iterations; and two-dimensional isogeny formulas alternate Hadamard transforms, squarings, and multiplications with different natural parallel widths. Consequently, a fast vectorized field multiplier alone does not imply a fast vectorized SQIsign implementation. The higher-level algorithms must be reorganized so that independent field operations are presented to the SIMD backend in sufficiently wide batches while avoiding frequent conversions between scalar and vector representations.

This observation motivates the central question of this work: *how much SIMD parallelism can be exposed inside the arithmetic dependency graphs of SQIsign, even when the outer algorithm is inherently sequential?* We address this question on x86-64 using AVX-512IFMA. Rather than treating AVX-512 as the contribution in itself, we use it as a concrete eight-lane realization of a broader scheduling methodology. When a primitive exposes enough independent arithmetic internally, we vectorize within the primitive; when it does not, we batch several points or evaluations; and when several related operations share inputs, we fuse them so that their combined dependency graph fills the available lanes. This perspective leads to vectorized schedules for xDBLADD, batched point doubling, biscalar multiplication, cubical-arithmetic pairings, and one- and two-dimensional isogenies.

The recent ARM implementation of SQIsign by De Feo *et al.* [FJWY26] also demonstrates that vectorization is effective for the curve-side part of *IdealToIsogeny*. It uses NEON finite-field arithmetic together with batched operations tailored to two-dimensional isogeny chains. Our work is complementary in both platform and scope: we study AVX-512IFMA on x86-64 and place particular emphasis on exposing parallelism inside scalar-multiplication ladders, pairing computations, and related elliptic-curve primitives, in addition to the higher-dimensional isogeny layer. Where our formulas overlap with the ARM work, we state this explicitly; where the scheduling differs, we describe the dependency structure that motivates our choice.

Contributions. Our contributions are as follows.

1. **Algorithm-level SIMD scheduling for SQIsign.** We reorganize the curve-side arithmetic of SQIsign to expose fine-grained parallelism beyond

the finite-field layer. In particular, we give SIMD schedules for projective xDBLADD without parameter normalization, simultaneous point doubling in x-only and modified Jacobian coordinates, the main loop of the two-dimensional biscalar ladder, fused cubical doubling/differential-addition steps for pairings, and batched one- and two-dimensional isogeny evaluation. A common design principle is to schedule the dependency graph in SIMD-width arithmetic rounds, using batching or fusion when one primitive alone does not provide sufficient width.

2. **End-to-end AVX-512IFMA implementation of SQIsign.** We implement these schedules using a uniform radix-2^{51} representation and AVX-512IFMA across all three NIST security levels. At the primitive level, eight-way batching improves the amortized throughput of 4-isogeny and $(2,2)$-isogeny evaluation by 4.64–5.47× and 4.38–5.31×, respectively, over the scalar C baseline. End-to-end, the resulting implementation achieves speedups of 1.76×, 1.71×, and 3.18× for Level-I key generation, signing, and verification. Our curve-side optimizations are structurally orthogonal to high-level changes such as Qlapoti [BCE+25]; the combined implementation confirms that the two can be used together, yielding Level-I key-generation and signing speedups of 2.90× and 2.69×.
3. **Cross-scheme validation on CORAL.** To test whether the vectorization methodology is specific to SQIsign, we apply the same AVX-512IFMA backend and applicable higher-dimensional schedules to CORAL [BBRS26], a recent restricted isogeny group action for post-quantum NIKE based on two-dimensional 2-isogenies. Across our five tested parameter sets, AVX-512 improves key generation by 1.28–1.40× and shared-key computation by 1.92–2.46× over the reference C implementation. For the `lvl5` parameter set, for which a Broadwell backend is also available, AVX-512 is 1.04× faster in key generation and 1.64× faster in shared-key computation. This second application provides end-to-end evidence that the same vectorization approach transfers to a distinct public-key primitive.

Related work. Cheng *et al.* [CFGR22] demonstrated highly vectorized SIKE implementations using AVX-512IFMA and established techniques for 52-bit IFMA-based Montgomery arithmetic that motivate our finite-field layer. De Feo *et al.* [FJWY26] subsequently presented the first vectorized SQIsign implementation for high-performance ARM processors, obtaining non-trivial end-to-end gains from NEON field arithmetic and batched curve-side operations. Our work should therefore not be viewed merely as replacing NEON with AVX-512: the main additional question studied here is how to restructure a broader set of SQIsign arithmetic dependency graphs—in particular ladders and pairings—to create sufficiently wide SIMD rounds.

Recent work also improves SQIsign along dimensions orthogonal to vectorization. Qlapoti [BCE+25] accelerates ideal-to-isogeny translation at the algorithmic level, while Superglue [Dup25] improves formulas for $(2,2)$-gluing isogenies. Kim *et al.* [KLKL25] replace GMP-dependent quaternion computations with fixed-precision integer arithmetic. Our implementation focuses on the curve-side

computation and can be combined with such changes; we evaluate the combination with Qlapoti explicitly. CORAL [BBRS26] uses two-dimensional 2-isogenies to evaluate a restricted isogeny group action and reports an unoptimized C implementation. Its substantially different high-level functionality, together with its overlap in higher-dimensional arithmetic, makes it a useful test case for the reusability of our SIMD techniques.

Outline. section 2 reviews SQIsign and the AVX-512IFMA instructions used in our implementation. section 3 describes the finite-field representation and the transition from field-level to algorithm-level vectorization. section 4 develops SIMD schedules for elliptic-curve operations and pairings, and section 5 treats one- and two-dimensional isogenies. section 6 evaluates both SQIsign and the CORAL case study. section 7 concludes.

2 Preliminaries

2.1 SQIsign

SQIsign is an isogeny-based candidate in the NIST post-quantum digital-signature standardization process and relies on the hardness of the endomorphism ring problem for supersingular elliptic curves, a mathematical foundation distinct from lattice-based and hash-based alternatives. At its core, the scheme instantiates a Fiat-Shamir transform over a sigma protocol proving knowledge of an isogeny path between supersingular curves.

Mathematical Foundation. Let p be a prime of cryptographic size, and consider the finite field $\mathbb{F}_{p^2}$. A supersingular elliptic curve E defined over $\mathbb{F}_{p^2}$ possesses the property that its endomorphism ring $\mathrm{End}(E)$ is isomorphic to a maximal order in a quaternion algebra $\mathcal{B}_{p,\infty}$ ramified at p and infinity. The Deuring correspondence establishes an equivalence between isogenies originating from a fixed base curve E_0 and left ideals of the corresponding maximal order $\mathcal{O}_0 \cong \mathrm{End}(E_0)$. This correspondence enables the translation of algebraic operations in quaternion algebras into geometric operations on elliptic curves, and vice versa.

Protocol Overview. The signature generation process comprises three phases: commitment, challenge, and response. The prover generates a secret isogeny $\varphi_{\mathrm{sk}} : E_0 \to E_{\mathrm{pk}}$ whose codomain constitutes the public key. During signing, the prover samples a random commitment isogeny $\varphi_{\mathrm{com}} : E_0 \to E_{\mathrm{com}}$. Upon receiving a challenge isogeny $\varphi_{\mathrm{chl}} : E_{\mathrm{pk}} \to E_{\mathrm{chl}}$ from the verifier, the prover must demonstrate knowledge of a response isogeny $\varphi_{\mathrm{rsp}} : E_{\mathrm{com}} \to E_{\mathrm{chl}}$ that does not backtrack through φ_{chl}. Crucially, the prover utilizes the Deuring correspondence to compute an ideal representation of the composite isogeny $\varphi_{\mathrm{chl}} \circ \varphi_{\mathrm{sk}} \circ \hat{\varphi}_{\mathrm{com}}$, then samples an equivalent ideal amenable to efficient isogeny translation.

Computational Components. The performance characteristics of SQIsign are dominated by the Ideal-to-Isogeny conversion routine, which comprises two distinct computational components. The first involves arithmetic over quaternion algebras, including ideal equivalence testing, lattice reduction, and norm computation. In the official SQIsign implementation evaluated in this work, these routines use the GNU Multiple Precision Arithmetic Library (GMP) for arbitrary-precision integer arithmetic, and we leave this algebraic component unchanged. Recent work by Kim *et al.* [KLKL25] shows that these quaternion operations can instead be implemented using fixed-precision integer arithmetic, eliminating the dependence on GMP.

The second component comprises the "curve-side" computations, which are the primary focus of this work. These include finite-field arithmetic in $\mathbb{F}_p$ and $\mathbb{F}_{p^2}$, elliptic-curve operations such as point doubling and scalar multiplication, and higher-dimensional isogeny evaluations over principally polarized abelian surfaces (PPAS). Contemporary instantiations of SQIsign employ two-dimensional isogeny chains derived from Kani's lemma [Kan97], necessitating extensive computations on Jacobian varieties and theta coordinates. The translation between ideal representations and these explicit geometric objects represents the primary bottleneck amenable to vectorization, as these operations ultimately reduce to modular multiplication and reduction in large prime fields.

Implementation Scope. Figure 1 illustrates the modular dependency structure of the SQIsign implementation. We employ the following color coding to indicate our optimization scope: **red** components comprise the GMP-based quaternion algebra operations used by the official SQIsign implementation evaluated in this work, which we leave unchanged; **green** components denote algorithms that have been redesigned for vectorized execution; and **blue** components represent higher-level operations that derive performance benefits from their dependencies on our vectorized field arithmetic primitives and vectorized algorithms.

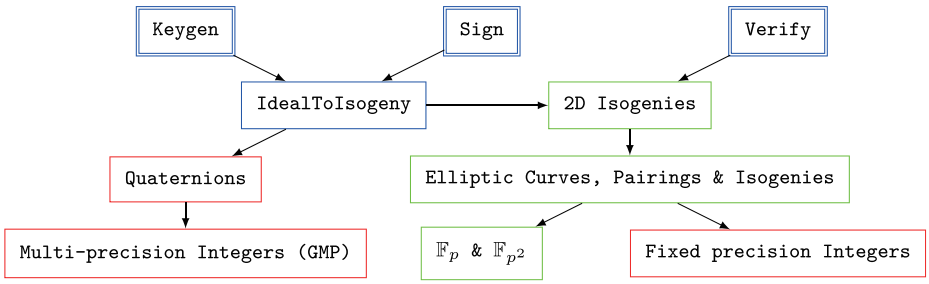


Fig. 1. SQIsign dependency graph, adapted from [FJWY26], with colors indicating our optimization scope.

2.2 AVX-512

The Advanced Vector Extensions 512 (AVX-512) extend the x86-64 SIMD architecture beyond AVX and AVX2. They provide wide vector operations that are well suited to high-throughput implementations of cryptographic primitives over large integer operands.

Architectural Overview. AVX-512 expands the x86-64 execution environment to encompass thirty-two 512-bit vector registers (ZMM0-ZMM31), effectively doubling both the register count and vector width compared to AVX2. The architecture supports operations on packed data elements ranging from 8-bit integers to 64-bit integers, enabling simultaneous processing of either eight 64-bit or sixteen 32-bit quantities within a single instruction. The AVX-512 Foundation (AVX-512F) provides the core integer and floating-point capabilities, while various sub-extensions address specific computational domains.

Integer Fused Multiply-Add (IFMA). Of particular relevance to cryptographic implementations is the AVX-512IFMA extension, introduced with the Cannon Lake microarchitecture and broadly available in Ice Lake and subsequent server-grade processors. It provides two specialized instructions, `vpmadd52luq` and `vpmadd52huq`. Both execute eight parallel 52-bit $\times$ 52-bit multiplications. The former adds the lower 52 bits of each product to the corresponding 64-bit destination lane, whereas the latter adds the upper 52 bits. This fused multiply-add capability reduces the instruction overhead of multi-precision Montgomery multiplication and modular reduction.

Hardware support and scope. AVX-512IFMA is available on several x86-64 processor families, including Intel server processors and recent AMD processors. Our implementation targets processors supporting AVX-512IFMA, and our experiments use an Intel Core i7-11700F.

3 Finite Field Arithmetic

The performance of isogeny-based cryptographic schemes, including SQIsign, is fundamentally constrained by arithmetic in the underlying finite fields. This section describes our AVX-512 implementation of the prime-field $\mathbb{F}_p$ and quadratic extension-field $\mathbb{F}_{p^2}$ operations. Our approach follows the vectorization strategies established in prior work on isogeny-based cryptography [CFGR22], utilizing the (8×1)-way parallelization pattern at the $\mathbb{F}_p$ layer.

3.1 Representation and Data Layout

The arithmetic framework of our implementation builds upon Montgomery's modular multiplication methodology [Mon85], which eliminates expensive division operations through a precomputed radix-dependent inverse. We adopt a

radix-2^{51} representation for field elements, motivated by the interplay between the 52-bit multiplier width of AVX-512IFMA and the requirement for lazy reduction. Field elements are decomposed into five 51-bit limbs, stored within the lower 52 bits of each 64-bit lane to prevent overflow during IFMA operations. The use of 51-bit limbs (rather than the full 52 bits) reserves headroom for carry accumulation during consecutive operations, thereby enabling lazy reduction strategies at higher algorithmic layers.

8-Way Limb Vector Organization. Our vectorized $\mathbb{F}_p$ implementation adopts an (8×1)-way parallelization strategy, processing eight independent field elements simultaneously within a single 512-bit ZMM register. Following the approach in [CFGR22], we assign each 64-bit lane to a distinct field element, maintaining spatial correspondence between the logical and physical representation.

Taking the NIST Level I parameter set as an example, given eight field elements $a^{(0)}, \ldots, a^{(7)} \in \mathbb{F}_p$, each represented as five 51-bit limbs, the limb vector set $\mathcal{V}$ is organized as:

$$\mathcal{V} = \left\{ \begin{array}{l} [a_0^{(0)}, a_0^{(1)}, a_0^{(2)}, a_0^{(3)}, a_0^{(4)}, a_0^{(5)}, a_0^{(6)}, a_0^{(7)}] \\ [a_1^{(0)}, a_1^{(1)}, a_1^{(2)}, a_1^{(3)}, a_1^{(4)}, a_1^{(5)}, a_1^{(6)}, a_1^{(7)}] \\ [a_2^{(0)}, a_2^{(1)}, a_2^{(2)}, a_2^{(3)}, a_2^{(4)}, a_2^{(5)}, a_2^{(6)}, a_2^{(7)}] \\ [a_3^{(0)}, a_3^{(1)}, a_3^{(2)}, a_3^{(3)}, a_3^{(4)}, a_3^{(5)}, a_3^{(6)}, a_3^{(7)}] \\ [a_4^{(0)}, a_4^{(1)}, a_4^{(2)}, a_4^{(3)}, a_4^{(4)}, a_4^{(5)}, a_4^{(6)}, a_4^{(7)}] \end{array} \right\}$$

where each row constitutes a 512-bit ZMM register containing the i-th limb of all eight elements.

3.2 Prime-Field $\mathbb{F}_p$ Operations

The special form of SQIsign's characteristic prime $p = c \cdot 2^f - 1$ (with small c) renders it amenable to optimized Montgomery reduction where the multiplicative factor involves only the small constant c [FLOR18]. This Montgomery-friendly property simplifies the modular reduction step.

Multiplication and Reduction. The modular multiplication leverages the special form $p = c \cdot 2^f - 1$ to streamline the reduction phase. For SQIsign's parameters, the Montgomery reduction of a 510-bit intermediate product proceeds via multiplication by the precomputed constant, implemented using a sequence of `vpmadd52luq` and `vpmadd52huq` instructions. The five-limb structure of our operands ensures that the reduction step requires only a minimal number of vector operations, with carries propagated across the 51-bit boundaries through mask-shift-add sequences.

3.3 Quadratic Extension $\mathbb{F}_{p^2}$ Layer

Operations in $\mathbb{F}_{p^2}$ are built upon the $\mathbb{F}_p$ primitives using the quadratic extension with irreducible polynomial $X^2 + 1$ (exploiting the fact that $p \equiv 3 \pmod 4$).

We implement three distinct parallelization strategies for $\mathbb{F}_{p^2}$ multiplication and squaring, corresponding to $(8 \times 1 \times 1)$-way, $(4 \times 2 \times 1)$-way, and $(2 \times 4 \times 1)$-way execution patterns, where the notation $(w \times x \times y)$ indicates w parallel $\mathbb{F}_{p^2}$ operations, each utilizing x parallel $\mathbb{F}_p$ operations, with y denoting the lane utilization at the lowest level.

- $(8 \times 1 \times 1)$**-way:** Eight independent $\mathbb{F}_{p^2}$ multiplications proceed in parallel, with each $\mathbb{F}_p$ operation internally utilizing the (8×1)-way limb organization. This mode is designed for high throughput on batch operations.
- $(4 \times 2 \times 1)$**-way:** Four $\mathbb{F}_{p^2}$ multiplications execute concurrently, with each multiplication internally processing two $\mathbb{F}_p$ operations (e.g., real and imaginary components) in parallel. This hybrid approach balances parallelism with register pressure for intermediate computations.
- $(2 \times 4 \times 1)$**-way:** Two $\mathbb{F}_{p^2}$ multiplications proceed with four-fold internal parallelism, suitable for algorithms requiring simultaneous processing of multiple field elements within a single higher-level operation.

The choice among these parallelization modes depends on the specific requirements of the higher-level algorithm. The $(8 \times 1 \times 1)$-way approach excels when processing independent batches of field elements, such as in simultaneous evaluation of isogenies at multiple points. The $(4 \times 2 \times 1)$-way and $(2 \times 4 \times 1)$-way variants are preferred when the algorithmic structure demands internal parallelism within individual $\mathbb{F}_{p^2}$ operations, such as in point addition formulas where real and imaginary components can be computed concurrently. Each implementation employs either schoolbook or Karatsuba decomposition at the coefficient level, selected based on the trade-off between multiplication count and addition/subtraction overhead under the specific parallelization constraints.

3.4 Addition and Subtraction

Vectorized implementation of $\mathbb{F}_{p^2}$ addition and subtraction is straightforward. The addition $r = a + b = (a_0 + b_0) + (a_1 + b_1)i$ in $\mathbb{F}_{p^2}$ consists of two independent additions in $\mathbb{F}_p$, which are executed in parallel using the `vpaddq` instruction. Similarly, subtraction employs the `vpsubq` instruction.

AVX-512 provides mask registers (k-registers) that enable selective operation on specific lanes of a ZMM register. By utilizing masked `vpaddq` and `vpsubq` instructions, we can perform addition on certain lanes while performing subtraction on others, deferring the modular correction until a unified reduction step. This capability allows us to mix addition and subtraction operations within the same algorithmic step (e.g., computing both $a + b$ and $a - b$ simultaneously, or handling mixed sequences of additions and subtractions that arise in algorithms such as the Hadamard operator) without requiring intermediate reductions or sign corrections.

3.5 Performance Evaluation

Table 1 reports the throughput of $\mathbb{F}_{p^2}$ multiplication and squaring under the NIST Level I parameter set ($p = 5 \cdot 2^{248} - 1$). All entries are reported as cycles

per $\mathbb{F}_{p^2}$ operation. For the x64 assembly and `modarith` implementations, this is simply the cycle count of one scalar operation. For the AVX-512IFMA variants, the reported value is the *amortized* cost per operation: we measure a SIMD batch containing eight, four, or two concurrent $\mathbb{F}_{p^2}$ operations and divide the batch cycle count by the corresponding batch size. For example, the 38-cycle entry for 8-way multiplication corresponds to approximately 304 cycles for a batch of eight independent multiplications. The comparison includes: (1) hand-optimized x64 assembly utilizing 64-bit limbs with `mulx`/`adcx`/`adox` instructions; (2) a C implementation generated by the `modarith` framework using 51-bit limbs but without vectorization; and (3) our AVX-512IFMA vectorized implementations with varying degrees of parallelism at the $\mathbb{F}_{p^2}$ layer.

Table 1. Throughput of $\mathbb{F}_{p^2}$ arithmetic for NIST Level I, in amortized cycles per operation. For AVX-512IFMA, the batch cycle count is divided by the number of concurrent $\mathbb{F}_{p^2}$ operations.

Operation	**x64 ASM**	`modarith`	**AVX-512IFMA**		
			8-way	**4-way**	**2-way**
Mul	102	246	38	46	51
Sqr	72	158	22	25	49

The results demonstrate substantial throughput improvements over both scalar variants. Relative to the optimized x64 assembly, the 8-way AVX-512IFMA implementation reduces the amortized cost from 102 to 38 cycles for multiplication (2.68×) and from 72 to 22 cycles for squaring (3.27×). The wider vectorization modes also improve amortized throughput: for example, the 2-way and 4-way squaring variants correspond to approximately 98 and 100 cycles per batch, respectively, while producing two and four results. Notably, the performance gap between the 51-bit non-vectorized C implementation and the 64-bit assembly baseline stems primarily from the difference in limb count: for the 248-bit prime p, the 64-bit representation requires only 4 limbs, whereas the 51-bit representation requires 5 limbs. This difference directly affects the number of elementary multiplications required by finite-field multiplication.

3.6 From Field-Level to Algorithm-Level Vectorization

Modern x86-64 processors already provide strong scalar multi-precision arithmetic through instructions such as `mulx`, `adcx`, and `adox`. A radix-2^{51} representation is therefore not automatically competitive when executed scalarly: it typically uses more limbs than a 64-bit representation and pays additional carry-management costs. The purpose of the radix-2^{51} representation is instead to provide a common data layout for IFMA and to keep data vectorized across higher-level operations.

This leads to a cross-layer design constraint. Vectorizing isolated $\mathbb{F}_{p^2}$ multiplications while repeatedly packing and unpacking data at elliptic-curve or isogeny boundaries can erase much of the SIMD gain. We therefore keep field elements in the vector representation through long stretches of the curve-side computation and organize higher-level formulas directly around the available SIMD width.

3.7 SIMD Scheduling Principles

We use three recurring transformations. *Intra-primitive scheduling* groups independent multiplications or squarings that already occur in one formula, as in xDBLADD. *Batching* combines several point or isogeny evaluations when a single instance does not expose enough independent operations. *Fusion* combines related operations with shared inputs, as in the cubical-arithmetic pairing step. A fourth design choice concerns *normalization*: a scalar formula with fewer multiplications is not necessarily the best SIMD formula if normalization itself is expensive or does not reduce the number of vector rounds.

Table 2 summarizes the main schedules developed in the following sections. The relevant metric is not only the total number of field operations but also the number of dependency-constrained multiplication/squaring rounds after scheduling. This is why several of our algorithms deliberately differ from scalar operation-count-minimal formulas.

4 Elliptic Curve Arithmetic

In this section, we review the fundamental elliptic curve operations underlying SQIsign, including point doubling, scalar multiplication, pairings, and discrete logarithm computations. For efficiency in isogeny arithmetic, SQIsign predominantly employs Montgomery curves of the form $BY^2Z = X^3 + AX^2Z + XZ^2$ equipped with projective x-only coordinates $(X : Z)$. The curve parameter A is also represented projectively as $A = (A_{\mathrm{Pr}} : C_{\mathrm{Pr}})$ with $A = A_{\mathrm{Pr}}/C_{\mathrm{Pr}}$, and A_{24} is represented projectively as $(A_{24} : C_{24}) = (A_{\mathrm{Pr}} + 2C_{\mathrm{Pr}} : 4C_{\mathrm{Pr}})$.

However, certain high-dimensional isogeny computations require access to the full (x, y) coordinates of points. In such cases, SQIsign transitions to Jacobian coordinates on Montgomery curves, or alternatively, to Modified Jacobian coordinates[CMO98] on short Weierstrass curves $y^2 = x^3+ax+b$. The coordinate transformations are defined as follows:

- For Jacobian coordinates $(X : Y : Z)$: $x = X/Z^2, y = Y/Z^3$
- For Modified Jacobian coordinates $(X : Y : Z : T)$: $x = X/Z^2, y = Y/Z^3, T = aZ^4$

The remainder of this section is organized as follows. subsection 4.1 presents vectorized implementations of point doubling across various coordinate models. subsection 4.2 turns to scalar multiplication, describing vectorized algorithms for both the Montgomery ladder and two-dimensional multi-scalar multiplication. subsection 4.3 addresses the vectorization of pairing operations employed in elliptic curve discrete logarithm computations.

Primitive	Parallelization source	SIMD scheduling strategy
xDBL	Intra-primitive (width 2)	Pair the independent products/squares; avoid normalization when it does not reduce vector rounds.
Three xDBLs	Cross-instance batching	Normalize once, then schedule three doublings together to fill four-way $\mathbb{F}_{p^2}$ operations.
Two modified-Jacobian DBLs	Cross-instance batching	Interleave the two dependency graphs and balance multiplication/squaring rounds.
xDBLADD	Intra-primitive (width 4)	Schedule doubling and differential addition jointly with projective $(A_{24} : C_{24})$.
Biscalar ladder iteration	Intra-iteration scheduling	Reorganize 18 field multiplications into five four-way multiplication rounds.
Cubical pairing step	Cross-operation fusion	Fuse one cubical doubling and two differential additions into three four-way multiplication rounds and one four-way squaring round.
1D/2D isogeny evaluation	Cross-instance batching	Evaluate several queued points together to amortize marshalling and saturate SIMD lanes.

Table 2. Recurring algorithm-level SIMD scheduling patterns used in our SQIsign implementation. Counts refer to the $\mathbb{F}_{p^2}$-level schedules described in the corresponding sections.

4.1 Point Doubling

We begin with point doubling in x-only coordinates. Given the curve parameter $(A_{24} : C_{24})$, a single doubling operation requires 2 squarings and 4 multiplications. Due to data dependencies, at most two multiplications or squarings can be executed simultaneously. In the reference implementation of SQIsign, a normalization procedure is invoked when more than 50 consecutive doublings are performed, reducing C_{24} to 1 and thereby decreasing the multiplication count per doubling to 3. However, this optimization offers no benefit for our vectorized implementation; consequently, we consistently omit the normalization step when performing vectorized point doublings.

During the signing and verification procedures of SQIsign, multiple steps require simultaneously doubling a full-order basis to adjust them to an appropriate order. This entails performing point doublings on three x-only coordinate points on the same curve, which exposes more cross-instance parallelism than a single doubling. Without normalization, doubling three points requires 6 squarings and 12 multiplications in total; after normalization, which reduces C_{24} to 1, the multiplication count decreases to 9. This reduction translates to a decrease from 5 to 4 multiplication rounds under four-way parallelization. Given that the number

Algorithm 1 Vectorized xDBL_A24

Input: A projective point $P = (X_P : Z_P)$ and the Montgomery coefficient $(A_{24} : C_{24})$ of the curve E.

Output: The projective point $[2]P = (X_{2P} : Z_{2P})$.

1: $t_0 \leftarrow X_P + Z_P$		$s_0 \leftarrow X_P - Z_P$
2: $t_1 \leftarrow t_0^2$		$s_1 \leftarrow s_0^2$
3: $t_2 \leftarrow t_1 - s_1$		
4: $t_3 \leftarrow t_2 \times A_{24}$		$s_3 \leftarrow s_1 \times C_{24}$
5: $t_4 \leftarrow t_3 + s_3$		
6: $t_5 \leftarrow s_3 \times t_1$		$s_5 \leftarrow t_2 \times t_4$
7: **return** $[2]P = (t_5 : s_5)$		

of doublings required for order adjustment of bases is typically substantial, we consistently perform the normalization step prior to executing the operation.

Algorithm 2 Doubling three points on the same curve after normalizing $C_{24} = 1$.

Input: Projective points $P = (X_P : Z_P), Q = (X_Q : Z_Q), R = (X_R : Z_R)$ and the normalized Montgomery coefficient A_{24} of the curve E, with $C_{24} = 1$.

Output: The projective points $[2]P = (X_{2P} : Z_{2P}), [2]Q = (X_{2Q} : Z_{2Q}), [2]R = (X_{2R} : Z_{2R})$.

1: $t_0 \leftarrow X_P + Z_P$	$s_0 \leftarrow X_P - Z_P$	$m_0 \leftarrow X_Q + Z_Q$	$n_0 \leftarrow X_Q - Z_Q$
2: $t_1 \leftarrow t_0^2$	$s_1 \leftarrow s_0^2$	$m_1 \leftarrow m_0^2$	$n_1 \leftarrow n_0^2$
3: $t_2 \leftarrow t_1 - s_1$	$s_2 \leftarrow m_1 - n_1$	$m_2 \leftarrow X_R + Z_R$	$n_2 \leftarrow X_R - Z_R$
4: $t_3 \leftarrow t_2 \times A_{24}$	$s_3 \leftarrow s_2 \times A_{24}$	$m_3 \leftarrow m_2 \times m_2$	$n_3 \leftarrow n_2 \times n_2$
5: $t_4 \leftarrow t_3 + s_1$	$s_4 \leftarrow s_3 + n_1$	$m_4 \leftarrow m_3 - n_3$	
6: $t_5 \leftarrow t_4 \times t_2$	$s_5 \leftarrow s_4 \times s_2$	$m_5 \leftarrow m_4 \times A_{24}$	
7: $t_6 \leftarrow m_5 + n_3$			
8: $t_7 \leftarrow t_1 \times s_1$	$s_7 \leftarrow m_1 \times n_1$	$m_7 \leftarrow m_3 \times n_3$	$n_7 \leftarrow m_4 \times t_6$
9: **return** $[2]P = (t_7 : t_5), [2]Q = (s_7 : s_5), [2]R = (m_7 : n_7)$			

The overwhelming majority of elliptic curve operations in SQIsign are performed using x-only coordinates on Montgomery curves. However, during higher-dimensional isogeny computations, complete projective coordinate arithmetic becomes occasionally necessary. The most common scenario arises in the computation of $(2, 2)$-isogeny chains, where iterative doublings of two points are required (see [AAA+24, Algorithm 8.46]). In the current SQIsign implementation, such operations are mostly accomplished by mapping points to modified Jacobian coordinates on short Weierstrass curves, with each point doubling incurring a cost of 3 finite field multiplications and 5 finite field squarings.

In Algorithm 3, we present a simultaneous doubling of two points with the multiplications and squarings organized into balanced vectorized operations. In [FJWY26], De Feo *et al.* analyze the same setting and adopt an alternative modified-Jacobian doubling formula requiring 4 finite-field multiplications and

4 finite-field squarings per point. The formula used here instead requires 3 multiplications and 5 squarings per point, so the two choices expose different multiplication/squaring mixes to the SIMD scheduler. We retain the formula used by the SQIsign code base and reorganize two simultaneous doublings into balanced SIMD rounds.

The schedule in Algorithm 3 is the standard modified-Jacobian doubling formula written so that two points share the same SIMD rounds. For each point, t_6 (respectively s_6) is the usual quantity $S = 4XY^2$, while t_4 (respectively s_4) is $M = 3X^2 + T$. Hence $X_{2P} = M^2 - 2S$ and $Y_{2P} = M(S - X_{2P}) - 8Y^4$; the updated modified-Jacobian auxiliary coordinate is $T_{2P} = 16Y^4T$. The resulting expressions are the standard modified-Jacobian doubling identities, written here in the temporary-variable notation used by our SIMD schedule.

Algorithm 3 Vectorized batched DBLW

Input: Modified Jacobian points $P = (X_P : Y_P : Z_P : T_P), Q = (X_Q : Y_Q : Z_Q : T_Q)$ for a Weierstrass curve.

Output: The doubled modified Jacobian points $[2]P, [2]Q$.

1:	$t_0 \leftarrow X_P^2$	$s_0 \leftarrow X_Q^2$	$m_0 \leftarrow Y_P^2$	$n_0 \leftarrow Y_Q^2$
2:	$t_1 \leftarrow t_0 + t_0$	$s_1 \leftarrow s_0 + s_0$	$m_1 \leftarrow m_0 + m_0$	$n_1 \leftarrow n_0 + n_0$
3:	$t_2 \leftarrow t_1 + t_0$	$s_2 \leftarrow s_1 + s_0$	$m_2 \leftarrow m_1 + X_P$	$n_2 \leftarrow n_1 + X_Q$
4:	$t_3 \leftarrow m_2^2$	$s_3 \leftarrow n_2^2$	$m_3 \leftarrow m_1^2$	$n_3 \leftarrow n_1^2$
5:	$t_4 \leftarrow t_2 + T_P$	$s_4 \leftarrow s_2 + T_Q$	$m_4 \leftarrow m_3 + m_3$	$n_4 \leftarrow n_3 + n_3$
6:	$t_5 \leftarrow t_3 - t_0$	$s_5 \leftarrow s_3 - s_0$		
7:	$t_6 \leftarrow t_5 - m_3$	$s_6 \leftarrow s_5 - n_3$		
8:	$t_7 \leftarrow t_4 \times t_4$	$s_7 \leftarrow s_4 \times s_4$	$m_7 \leftarrow Y_P \times Z_P$	$n_7 \leftarrow Y_Q \times Z_Q$
9:	$t_8 \leftarrow t_7 - t_6$	$s_8 \leftarrow s_7 - s_6$		
10:	$t_9 \leftarrow t_8 - t_6$	$s_9 \leftarrow s_8 - s_6$		
11:	$t_{10} \leftarrow t_6 - t_9$	$s_{10} \leftarrow s_6 - s_9$		
12:	$t_{11} \leftarrow t_4 \times t_{10}$	$s_{11} \leftarrow s_4 \times s_{10}$	$m_{11} \leftarrow m_4 \times T_P$	$n_{11} \leftarrow n_4 \times T_Q$
13:	$t_{12} \leftarrow t_{11} - m_4$	$s_{12} \leftarrow s_{11} - n_4$		
14:	$t_{13} \leftarrow m_7 + m_7$	$s_{13} \leftarrow n_7 + n_7$	$m_{13} \leftarrow m_{11} + m_{11}$	$n_{13} \leftarrow n_{11} + n_{11}$
15:	**return** $[2]P = (t_9 : t_{12} : t_{13} : m_{13}), [2]Q = (s_9 : s_{12} : s_{13} : n_{13})$			

4.2 Scalar Multiplication

Scalar multiplication constitutes a fundamental operation employed across various subroutines within the SQIsign signature scheme. The SQIsign specification categorizes scalar multiplication algorithms into three distinct classes, all of which operate on Montgomery curves utilizing projective x-only coordinate representations:

Ladder The classical Montgomery ladder algorithm, originally introduced by Montgomery in [Mon87]. Given a point P on an elliptic curve E, this algorithm computes the scalar multiple $[m]P$.

Ladder3pt The three-point ladder algorithm proposed by Faz-Hernández et al. in [FLOR18]. Given points P, Q, and $P - Q$ on an elliptic curve E, this algorithm computes $P + [m]Q$.

LadderBiscalar The two-dimensional ladder algorithm due to Bernstein [Ber06]. Given points P, Q, and $P-Q$ on an elliptic curve E, this algorithm computes the biscalar multiple $[m]P + [n]Q$.

For the sake of brevity, we refrain from presenting the detailed theoretical foundations underlying these algorithms. Instead, we excerpt the (partial) pseudocode specifications as documented in the SQIsign reference implementation.

Algorithm 4 Ladder(P, E, m)

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

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

Algorithm 5 Ladder3pt$(P, Q, P - Q, (A_{24} : C_{24}), m)$

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

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

A careful examination of the Ladder and Ladder3pt algorithms reveals that their main loops exhibit strict data iteration dependencies, preventing parallel unrolling at the loop level; consequently, their computational performance is entirely determined by the execution efficiency of the underlying xDBLADD function. Therefore, the key to enhancing algorithmic efficiency through vectorization lies in exploiting the internal parallelism within the xDBLADD operation.

The xDBLADD function constitutes a combined operation that simultaneously performs point doubling and point addition. It takes as input the curve parameters together with points P, Q, and $P - Q$, and produces as output the values $P + Q$ and $[2]P$. In the literature [CFGR22], Cheng et al. conducted a comparative analysis of four-way vectorization strategies for the xDBLADD procedure as described in [CS09,NS22,HEY22], concluding that the vectorization approaches presented in [CS09] and [NS22] are heavily dependent on the special form of the Montgomery curve parameter A_{24}—a characteristic commonly encountered in classical elliptic curve cryptography—rendering them unsuitable for scalar multiplication implementations in isogeny-based cryptography. By contrast, the strategy proposed in [HEY22] was deemed more appropriate for this context.

However, the algorithm in [HEY22] relies on the curve parameter A, whereas in isogeny-based cryptography, curve parameters are typically expressed in the form $(A : C)$ or $(A_{24} : C_{24})$ for computational efficiency. To address this limitation, we present a redesigned four-way vectorization strategy for xDBLADD in Algorithm 6. This implementation imposes no special requirements on the parameter format and eliminates the need for parameter normalization operations; the parameters need only be provided in the $(A_{24} : C_{24})$ representation. To facilitate comparison with [NS22] and [HEY22], we additionally provide a graphical illustration of the algorithm in Figure 2.

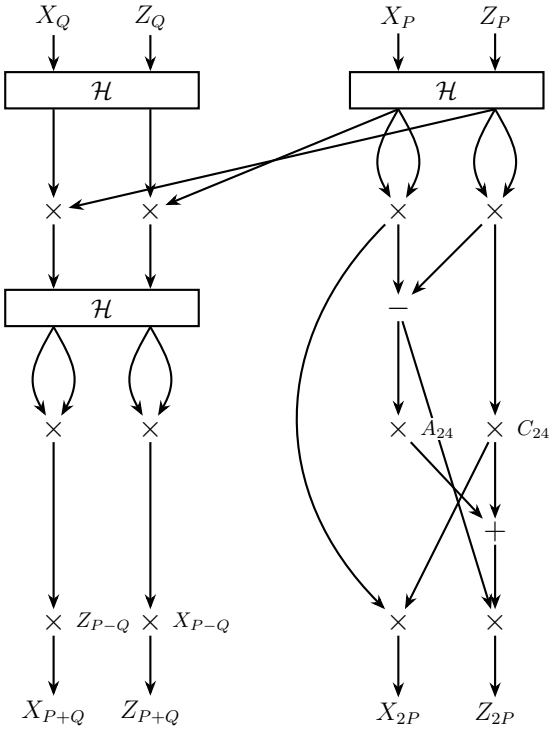
Algorithm 6 xDBLADD algorithm with 4-way parallelization at $\mathbb{F}_{p^2}$-level.

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

1:	$t_0 \leftarrow X_Q + Z_Q$	$s_0 \leftarrow X_Q - Z_Q$	$m_0 \leftarrow X_P + Z_P$	$n_0 \leftarrow X_P - Z_P$
2:	$t_1 \leftarrow t_0 \times n_0$	$s_1 \leftarrow s_0 \times m_0$	$m_1 \leftarrow m_0 \times m_0$	$n_1 \leftarrow n_0 \times n_0$
3:	$t_2 \leftarrow t_1 + s_1$	$s_2 \leftarrow t_1 - s_1$	$m_2 \leftarrow m_1 - n_1$	
4:	$t_3 \leftarrow t_2 \times t_2$	$s_3 \leftarrow s_2 \times s_2$	$m_3 \leftarrow m_2 \times A_{24}$	$n_3 \leftarrow n_1 \times C_{24}$
5:	$t_4 \leftarrow m_3 + n_3$			
6:	$t_5 \leftarrow t_3 \times Z_{P-Q}$	$s_5 \leftarrow s_3 \times X_{P-Q}$	$m_5 \leftarrow m_1 \times n_3$	$n_5 \leftarrow m_2 \times t_4$
7:	**return** $([2]P, P + Q) = \big((m_5 : n_5), (t_5 : s_5)\big)$			

We now proceed to analyze the LadderBiscalar algorithm. Owing to its substantially greater complexity compared to the aforementioned ladder algorithms, we present in Algorithm 7 only the core portion subject to our primary optimiza-

Fig. 2. Our vectorized strategy for xDBLADD



tions; the complete algorithmic specification may be found in Algorithm 8.8 in [AAA+24].

To facilitate comprehension, we begin with a brief overview of the algorithmic workflow. The fundamental distinction between this algorithm and its predecessors lies in the scalar processing mechanism: whereas previous ladder algorithms process input scalars bit-by-bit, with each loop iteration governed solely by the current bit value, LadderBiscalar exhibits significantly more intricate behavior. The operations performed in each loop iteration depend not merely on the "current bit," but additionally on the operation executed in the preceding iteration as well as the subsequent bit. This renders LadderBiscalar a "stateful" algorithm in a meaningful sense.

Consequently, the algorithm naturally decomposes into two distinct phases: the *Recoding stage* and the *Evaluation stage*. During the Recoding stage, the algorithm scans the input scalars m and n to determine the specific operation to be executed in each loop iteration. Subsequently, the Evaluation stage commences with the elliptic curve computations. The initialization procedure employs the points $(1 : 0)$, P, and Q—in a specific order—to establish the values R_0, R_1, and R_2, while simultaneously initializing four difference points D_1, D_2, F_1, and F_2.

Throughout the iterative process, the values R_0, R_1, and R_2 undergo transformation, whereas the difference points remain invariant.

The computational workload of each loop iteration consists of selecting one point from $\{R_0, R_1, R_2\}$ for point doubling, together with two differential additions performed on pairs of points. This computation encompasses a total of 18 finite field multiplications. In Algorithm 8, we reorganize this main loop into five four-way parallel multiplication operations. As normalization of parameters offers no reduction in the number of parallel multiplications, we maintain our design choice of parameter non-normalization.

Algorithm 7 LadderBiscalar$(P, Q, P-Q, (A_{24} : C_{24}), m, n)$ (Partial)

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

Output: The projective point $[m]P + [n]Q = (X_{[m]P+[n]Q} : Z_{[m]P+[n]Q})$.

29: **for** i **from** $k-1$ **down to** 0 **do**
30: $h \leftarrow r_{2i} + r_{2i+1}$
31: $T_0 \leftarrow R_{h \pmod 2}$
32: $T = (T_0, T_1) \leftarrow (T_0, R_2)$
33: $T_0 \leftarrow \text{xDBL}(T_{\lfloor h/2 \rfloor}, (A_{24} : C_{24}))$
34: $T_1 \leftarrow R_{r_{2i+1}}$
35: $T_2 \leftarrow R_{r_{2i+1}+1}$
36: **if** $r_{2i+1} = 1$ **then**
37: $TMP \leftarrow D_1$
38: $D_1 \leftarrow D_2$
39: $D_2 \leftarrow TMP$
40: $T_1 \leftarrow \text{xADD}(T_1, T_2, D_1)$
41: $T_2 \leftarrow \text{xADD}(R_0, R_2, F_1)$
42: **if** $h \pmod 2 = 1$ **then**
43: $TMP \leftarrow F_1$
44: $F_1 \leftarrow F_2$
45: $F_2 \leftarrow TMP$
46: $R_0 \leftarrow T_0$
47: $R_1 \leftarrow T_1$
48: $R_2 \leftarrow T_2$
49: $(X_{[m]P+[n]Q} : Z_{[m]P+[n]Q}) \leftarrow R_{(m \pmod 2)\oplus 1)+(n \pmod 2)\oplus 1)}$
50: **return** $[m]P + [n]Q = (X_{[m]P+[n]Q} : Z_{[m]P+[n]Q})$

4.3 Pairing and Discrete Logarithm

To accelerate computations, the key generation and signing procedures in SQIsign require the generation of a 2^e-torsion basis (Q_1, Q_2) on a specific elliptic curve, which is subsequently recorded within the private key or signature. To conserve

Algorithm 8 LadderBiscalar main loop algorithm with 4-way parallelization at $\mathbb{F}_{p^2}$-level.

Input: Projective points $(X_{T_0} : Z_{T_0}), (X_{T_1} : Z_{T_1}), (X_{T_2} : Z_{T_2}), (X_{R_0} : Z_{R_0}), (X_{R_2} : Z_{R_2}), (X_{D_1} : Z_{D_1}), (X_{F_1} : Z_{F_1})$.
Output: Projective points $R_0 = [2]T_0, R_1 = T_1 + T_2, R_2 = R_0 + R_2$.

1:	$t_0 \leftarrow X_{T_2} + Z_{T_2}$	$s_0 \leftarrow X_{T_2} - Z_{T_2}$	$m_0 \leftarrow X_{T_1} + Z_{T_1}$	$n_0 \leftarrow X_{T_1} - Z_{T_1}$
2:	$t_1 \leftarrow X_{R_0} + Z_{R_0}$	$s_1 \leftarrow X_{R_0} - Z_{R_0}$	$m_1 \leftarrow X_{R_2} + Z_{R_2}$	$n_1 \leftarrow X_{R_2} - Z_{R_2}$
3:	$t_2 \leftarrow s_0 \times m_0$	$s_2 \leftarrow t_0 \times n_0$	$m_2 \leftarrow t_1 \times n_1$	$n_2 \leftarrow s_1 \times m_1$
4:	$t_3 \leftarrow t_2 + s_2$	$s_3 \leftarrow t_2 - s_2$	$m_3 \leftarrow m_2 + n_2$	$n_3 \leftarrow m_2 - n_2$
5:	$t_4 \leftarrow t_3 \times t_3$	$s_4 \leftarrow s_3 \times s_3$	$m_4 \leftarrow m_3 \times m_3$	$n_4 \leftarrow n_3 \times n_3$
6:	$t_5 \leftarrow X_{T_0} + Z_{T_0}$	$s_5 \leftarrow X_{T_0} - Z_{T_0}$		
7:	$t_6 \leftarrow t_4 \times Z_{D_1}$	$s_6 \leftarrow s_4 \times X_{D_1}$	$m_6 \leftarrow t_5 \times t_5$	$n_6 \leftarrow s_5 \times s_5$
8:	$t_7 \leftarrow m_6 - n_6$			
9:	$t_8 \leftarrow t_7 \times A_{24}$	$s_8 \leftarrow n_6 \times C_{24}$	$m_8 \leftarrow m_4 \times Z_{F_1}$	$n_8 \leftarrow n_4 \times X_{F_1}$
10:	$t_9 \leftarrow t_8 + s_8$			
11:	$t_{10} \leftarrow m_6 \times s_8$	$s_{10} \leftarrow t_7 \times t_9$		
12:	**return** $R_0 = (t_{10} : s_{10}), R_1 = (t_6 : s_6), R_2 = (m_8 : n_8)$			

bandwidth, SQIsign does not store the coordinates of the basis points directly. Instead, the scheme proceeds from a deterministic basis (P_1, P_2) of $E[2^e]$ and computes the change-of-basis matrix

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

This matrix could theoretically be generated via discrete logarithm methods on the elliptic curve; however, for computational efficiency, SQIsign adopts an alternative approach involving the computation of five pairings followed by discrete logarithms in the finite field. We excerpt this algorithm in Algorithm 9.

Algorithm 9 ChangeOfBasis$_{2^e}(E, (P_1, P_2), (Q_1, Q_2))$

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

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

The current implementation of SQIsign employs the Tate-Lichtenbaum pairing to facilitate discrete logarithm computations. Notably, the pairing computation in SQIsign does not follow the classical Miller loop algorithm; instead, it utilizes an approach based on cubical arithmetic.

The theoretical foundations of cubical arithmetic trace back to reinterpretations of the Tate pairing as monodromy in biextensions associated to principal polarizations, as developed in [Sta08] and subsequently refined in [Rob24] through the lens of cubical arithmetic structures.

We refrain from presenting the complete theoretical framework of cubical arithmetic herein, confining our discussion to the core algorithmic steps relevant to SQIsign. The implementation employs level-2 cubical arithmetic associated to the divisor $2(0_E)$. To compute $t_n(P, Q)$, one first obtains cubical points $\widetilde{P}$ and $\widetilde{Q}$ above P and Q. This initialization is straightforward: the points are normalized to yield $\widetilde{P} = (X(P)/Z(P) : 1)$ and $\widetilde{Q} = (X(Q)/Z(Q) : 1)$. Subsequently, the CubicalLadder algorithm is executed to compute the cubical points $2^e\widetilde{P}$ and $2^e\widetilde{P} + \widetilde{Q}$. The CubicalLadder algorithm constitutes essentially a cubical arithmetic adaptation of the three-point ladder, wherein CubicalDiffAdd and CubicalDbl denote the cubical arithmetic variants of differential addition and point doubling, respectively. The detailed specifications appear in [AAA+24, Algorithms 8.13 and 8.14].

Algorithm 10 CubicalLadder$(E, e, \widetilde{P+Q}, \widetilde{P}, x(Q))$

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

Output: The cubical points $2^e\widetilde{P}$ and $2^e\widetilde{P} + \widetilde{Q}$.

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

Both differential addition and point doubling in cubical arithmetic require five finite field multiplications, which we reorganize into three four-way parallel vectorized multiplication operations. Pope et al. [PRR+25] observed that SQIsign invariably computes two bilinear pairings with respect to the same point (cf. Algorithm 9, Line 2), thereby enabling the reuse of intermediate values. Capitalizing on this insight, we fuse one point doubling with two differential additions to obtain Algorithm 11, which requires three four-way parallel multiplications and one four-way parallel squaring operation. The resulting ladder implementation can simultaneously generate the two bilinear pairings required in Algorithm 9.

Remark. The execution of the CubicalDbl algorithm necessitates normalized curve parameters A_{24}. One might observe that CubicalDbl is in fact identical to the x-only coordinate doubling formula on Montgomery curves, and consequently

Algorithm 11 cubicalDBLADDADD

Input: Three cubical points $\widetilde{P} = (X_P, Z_P), \widetilde{Q} = (X_Q, Z_Q), \widetilde{R} = (X_R, Z_R)$, the inverses of the differences $ixPR = 1/x(P - R), ixQR = 1/x(Q - R)$ and the normalized Montgomery coefficient A_{24} of the curve E.

Output: The cubical double $2\widetilde{R}$ and the cubical differential additions $\widetilde{P + R}$,$\widetilde{Q + R}$

1:	$t_1 \leftarrow X_P + Z_P$	$s_1 \leftarrow X_P - Z_P$	$m_1 \leftarrow X_R + Z_R$	$n_1 \leftarrow X_R - Z_R$
2:	$t_2 \leftarrow X_Q + Z_Q$	$s_2 \leftarrow X_Q - Z_Q$		
3:	$t_3 \leftarrow m_1 \times m_1$	$s_3 \leftarrow n_1 \times n_1$	$m_3 \leftarrow m_1 \times s_2$	$n_3 \leftarrow n_1 \times t_2$
4:	$t_4 \leftarrow t_3 - s_3$	$s_4 \leftarrow m_3 + n_3$	$m_4 \leftarrow m_3 - n_3$	
5:	$t_5 \leftarrow t_1 \times n_1$	$s_5 \leftarrow s_1 \times m_1$	$m_5 \leftarrow A_{24} \times t_4$	
6:	$t_6 \leftarrow t_5 + s_5$	$s_6 \leftarrow t_5 - s_5$	$m_6 \leftarrow s_3 + m_5$	
7:	$t_7 \leftarrow t_6 \times t_6$	$s_7 \leftarrow s_6 \times s_6$	$m_7 \leftarrow s_4 \times s_4$	$n_7 \leftarrow m_4 \times m_4$
8:	$t_8 \leftarrow ixPR \times t_7$	$s_8 \leftarrow t_3 \times s_3$	$m_8 \leftarrow m_6 \times t_4$	$n_8 \leftarrow ixQR \times m_7$
9:	**return** $2\widetilde{R} = (s_8 : m_8)$,$\widetilde{P + R} = (t_8 : s_7)$,$\widetilde{Q + R} = (n_8 : n_7)$			

surmise that the normalization operation could be eliminated by applying our previously established technique—introducing a multiplication by C_{24} within Algorithm 11—without increasing the number of vectorized multiplications while saving one field inversion. This approach, however, proves untenable. Such a modification to the doubling formula alters the normalization of $\widetilde{0}$, which in turn necessitates corresponding adjustments to the cubical differential addition formula; continued use of the original formulae would yield erroneous results. A comprehensive analysis of this phenomenon is provided in [Rob24, Section 5.2].

5 Isogenies in Dimension 1 and 2

5.1 Isogenies between Elliptic Curves

The evaluation of 4-isogenies on Montgomery curves follows the formulas established by Costello *et al.* [CLN16] and subsequently optimized for vectorization by Cheng *et al.* [CFGR22]. Given a kernel point $P_4 = (X_4 : Z_4)$ of order 4 and a point $P = (X_P : Z_P)$ on the domain curve, the image point $P' = \phi_4(P) = (X_{P'} : Z_{P'})$ is computed via the relations:

$$X_{P'} = 16 \cdot \left[(X_4X_P - Z_4Z_P)^2 + Z_4^2(X_P^2 - Z_P^2)\right] \cdot (X_4X_P - Z_4Z_P)^2,$$
$$Z_{P'} = 16 \cdot \left[(X_4Z_P - Z_4X_P)^2 - Z_4^2(X_P^2 - Z_P^2)\right] \cdot (X_4Z_P - Z_4X_P)^2.$$

Algebraically, these formulas expose at most two independent multiplications that can execute simultaneously—specifically, the computation of $X_4X_P \pm Z_4Z_P$ and $X_4Z_P \pm Z_4X_P$ followed by their squares. This structural constraint limits the theoretical parallelism per point evaluation to two-way, as recognized in prior work [CFGR22].

Cheng *et al.* [CFGR22] observed that isogeny chain computations using optimal strategy traversal naturally maintain a stack of points awaiting evaluation at each step. Rather than processing these points sequentially with repeated

scalar-to-vector data marshalling between steps, they demonstrated that batching multiple evaluations into a single vectorized invocation amortizes instruction overhead and improves throughput. We adopt this batching methodology in our SQIsign implementation, interleaving coordinates from multiple points across AVX-512 lanes to saturate the SIMD multipliers.

Table 4 quantifies this effect, demonstrating that processing multiple points concurrently yields speedups, despite the two-way structural constraint inherent to the formulas.

5.2 Isogenies between Principally Polarized Abelian Surfaces

SQIsign's two-dimensional isogeny computations operate on theta coordinates, requiring intensive point doubling operations on the PPAS. The theta model, originating from Mumford's seminal work on algebraic theta functions [Mum83], provides efficient arithmetic on principally polarized abelian varieties through level-2 theta constants. Lubicz and Robert [LR12] subsequently developed computational frameworks for isogeny evaluation within this coordinate system, which we adapt for our vectorized implementation. We provide vectorized implementations for both the precomputation phase (Algorithm 12) and the doubling operation (Algorithm 13), reorganizing the Hadamard transforms and coordinate multiplications to maximize SIMD lane utilization.

Algorithm 12 ThetaPrecomp_vec(0_A)

Input: Theta null point $0_A := (a : b : c : d)$.
Output: Auxiliary constants `consts` used for arithmetic.

1:	$t_0 \leftarrow a^2$	$s_0 \leftarrow b^2$	$m_0 \leftarrow c^2$	$n_0 \leftarrow d^2$
2:	$t_1 \leftarrow t_0 + s_0$	$s_1 \leftarrow t_0 - s_0$	$m_1 \leftarrow m_0 + n_0$	$n_1 \leftarrow m_0 - n_0$
3:	$t_2 \leftarrow t_1 + m_1$	$s_2 \leftarrow t_1 - m_1$	$m_2 \leftarrow s_1 + n_1$	$n_2 \leftarrow s_1 - n_1$
4:	$t_3 \leftarrow s_2 \cdot n_2$	$s_3 \leftarrow c \cdot d$	$m_3 \leftarrow t_2 \cdot m_2$	$n_3 \leftarrow a \cdot b$
5:	$abc \leftarrow n_3 \cdot c$	$abd \leftarrow n_3 \cdot d$	$acd \leftarrow s_3 \cdot a$	$bcd \leftarrow s_3 \cdot b$
6:	$ABC \leftarrow m_3 \cdot s_2$	$ABD \leftarrow m_3 \cdot n_2$	$ACD \leftarrow t_3 \cdot t_2$	$BCD \leftarrow t_3 \cdot m_2$
7:	`consts`$\leftarrow \{abc, abd, acd, bcd, ABC, ABD, ACD, BCD\}$			
8:	**return** `consts`			

For the generic evaluation of $(2, 2)$-isogenies, our derived formulas align with the vectorized approach presented in the ARM implementation [FJWY26, Algorithm 16]. Following the methodology established in the one-dimensional case, we apply the same batching principle to two-dimensional isogeny evaluations, processing multiple theta points concurrently to amortize data movement costs. The performance characteristics for these operations are summarized in Table 5.

Algorithm 13 ThetaDBL_vec(P,`consts`)

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

$x_P, y_P, z_P, w_P \leftarrow P$
$c_1, c_2, c_3, c_4, c_5, c_6, c_7, c_8 \leftarrow$ `consts`

$t_0 \leftarrow x_P^2$	$s_0 \leftarrow y_P^2$	$m_0 \leftarrow z_P^2$	$n_0 \leftarrow w_P^2$
$t_1 \leftarrow t_0 + s_0$	$s_1 \leftarrow t_0 - s_0$	$m_1 \leftarrow m_0 + n_0$	$n_1 \leftarrow m_0 - n_0$
$t_2 \leftarrow t_1 + m_1$	$s_2 \leftarrow t_1 - m_1$	$m_2 \leftarrow s_1 + n_1$	$n_2 \leftarrow s_1 - n_1$
$t_3 \leftarrow t_2^2$	$s_3 \leftarrow m_2^2$	$m_3 \leftarrow s_2^2$	$n_3 \leftarrow n_2^2$
$t_4 \leftarrow t_3 \cdot c_8$	$s_4 \leftarrow s_3 \cdot c_7$	$m_4 \leftarrow m_3 \cdot c_6$	$n_4 \leftarrow n_3 \cdot c_5$
$t_5 \leftarrow t_4 + s_4$	$s_5 \leftarrow t_4 - s_4$	$m_5 \leftarrow m_4 + n_4$	$n_5 \leftarrow m_4 - n_4$
$t_6 \leftarrow t_5 + m_5$	$s_6 \leftarrow t_5 - m_5$	$m_6 \leftarrow s_5 + n_5$	$n_6 \leftarrow s_5 - n_5$
$X_{2P} \leftarrow t_6 \cdot c_4$	$Y_{2P} \leftarrow m_6 \cdot c_3$	$Z_{2P} \leftarrow s_6 \cdot c_2$	$W_{2P} \leftarrow n_6 \cdot c_1$

return $(X_{2P} : Y_{2P} : Z_{2P} : W_{2P})$

6 Experimental Results

6.1 Experimental Setup

We implemented our vectorized SQIsign using AVX-512IFMA atop the official reference implementation[1]. The evaluated reference configuration uses the original GMP-based quaternion-arithmetic path; we do not use or modify the fixed-precision quaternion implementation of Kim *et al.* [KLKL25]. All SQIsign benchmarks were conducted on an Intel Core i7-11700F CPU (2.50 GHz base frequency) running Ubuntu 24.04 LTS with GCC 13.2.0. Hyper-threading and Turbo Boost were disabled. Unless explicitly labeled `modarith`, every "C" entry in the performance tables refers to the unmodified official SQIsign C reference implementation. The "ASM" entries denote the Broadwell-optimized x86-64 backend distributed with SQIsign, which uses hand-written 64-bit finite-field arithmetic.

Our AVX-512IFMA implementation uses radix-2^{51} field elements uniformly across the three NIST parameter sets. Since the `modarith`[2] generator uses different limb sizes for some parameter sets, we modified the generation scripts to emit 51-bit-limb scalar code for all levels. This representation-compatible scalar code is used only in the finite-field microbenchmarks of Table 1; the end-to-end and higher-level operation baselines remain the unmodified official C implementation.

Each benchmark is preceded by a warm-up phase and then executed 10,000 times on a pinned CPU core. Cycle counts are obtained using `RDTSC`, and we report the median over the 10,000 measurements.

Correctness validation. We validated each vectorized primitive against the corresponding scalar reference routine on randomized inputs for all supported param-

[1] `https://github.com/SQIsign/the-sqisign`
[2] `https://github.com/mcarrickscott/modarith`

eter sets. We also performed end-to-end tests of SQIsign key generation, signing, and verification, and checked shared-key agreement for the CORAL implementation.

Side-channel scope. The reference SQIsign implementation is not constant-time as a whole, primarily because of its quaternion-side computation. The vectorization transformations introduced here do not add secret-dependent branches, memory accesses, or SIMD mask patterns beyond those already present in the reference computation; lane scheduling and mask selection are determined by the algorithmic structure rather than secret data. We do not claim resistance against implementation-level side-channel attacks.

6.2 SQIsign Microbenchmarks

Table 3 reports core elliptic-curve and pairing-related operations at NIST security levels I, III, and V. The "Iter." column denotes the number of doublings for xDBL routines, the scalar bit length for ladder routines, and the exponent of the pairing level (that is, $2^{\text{Iter.}}$-torsion) for pairing computations. These values reflect the workloads used by the corresponding SQIsign parameter sets. Speedups in this table are relative to the reference C implementation, while the ASM rows show how the vectorized schedules compare with an optimized scalar x86-64 implementation.

Table 3: Elliptic curve operation performance (point doublings, ladders, discrete logarithms) by NIST level. Speedups are relative to the C reference implementation.

Algorithm	Level	Iter.	Impl.	Cycles	Speedup
`ec_dbl_iter`	I	120	C	169938	1.00×
			ASM	65198	2.61×
			AVX-512	59046	2.88×
	III	182	C	397056	1.00×
			ASM	222310	1.79×
			AVX-512	157548	2.52×
	V	245	C	662712	1.00×
			ASM	425704	1.56×
			AVX-512	289502	2.29×

Continued on next page

Table 3 – continued from previous page

Algorithm	Level	Iter.	Impl.	Cycles	Speedup
ec_dbl_iter_basis	I	120	C	428414	1.00×
			ASM	195574	2.19×
			AVX-512	108278	3.96×
	III	182	C	1061930	1.00×
			ASM	666676	1.59×
			AVX-512	317378	3.35×
	V	245	C	1993704	1.00×
			ASM	1279354	1.56×
			AVX-512	602438	3.31×
DBLW	I	123	C	530512	1.00×
			ASM	240406	2.21×
			AVX-512	117354	4.52×
	III	186	C	1279838	1.00×
			ASM	748552	1.71×
			AVX-512	328336	3.90×
	V	248	C	2590720	1.00×
			ASM	1409952	1.84×
			AVX-512	602044	4.30×
ec_mul	I	128	C	345570	1.00×
			ASM	146224	2.36×
			AVX-512	94644	3.65×
	III	194	C	874340	1.00×
			ASM	526508	1.66×
			AVX-512	277738	3.15×
	V	255	C	1581856	1.00×
			ASM	982576	1.61×
			AVX-512	512496	3.09×
ec_ladder3pt	I	256	C	726784	1.00×
			ASM	297622	2.44×
			AVX-512	193676	3.75×
	III	384	C	1770924	1.00×
			ASM	1073598	1.65×
			AVX-512	563822	3.14×
	V	512	C	3289162	1.00×
			ASM	2049516	1.60×
			AVX-512	1049416	3.13×

Continued on next page

Table 3 – continued from previous page

Algorithm	Level	Iter.	Impl.	Cycles	Speedup
ec_biscalar_mul	I	248	C	1161592	1.00×
			ASM	499962	2.32×
			AVX-512	321898	3.61×
	III	376	C	2855692	1.00×
			ASM	1717500	1.66×
			AVX-512	925660	3.09×
	V	500	C	5248922	1.00×
			ASM	3265876	1.61×
			AVX-512	1743904	3.01×
tate_dlog_partial	I	248	C	4288454	1.00×
			ASM	1907992	2.25×
			AVX-512	1238384	3.46×
	III	376	C	11149732	1.00×
			ASM	6796450	1.64×
			AVX-512	3872494	2.88×
	V	500	C	22518594	1.00×
			ASM	13192902	1.71×
			AVX-512	7468538	3.02×

AVX-512 also consistently outperforms the optimized scalar ASM backend across all tested primitives and parameter sets in Table 3. The improvement over ASM ranges from 1.10× to 2.34×. The largest gains occur in operations that expose wider parallelism, such as batched point doubling, while dependency-heavy ladder and pairing routines still retain substantial improvements. Thus, the gains are not limited to comparisons against portable C code.

6.3 Isogeny-Evaluation Throughput

Tables 4 and 5 isolate the batching effect for one- and two-dimensional isogeny evaluation. The "Cyc./Inst." column reports amortized cycles per evaluated instance. Increasing the batch size generally lowers this cost because more independent field operations can be scheduled in the same vector rounds and the cost of data marshalling is amortized across more points. The scaling is not strictly monotone for every intermediate batch size: for $(2, 2)$-isogeny evaluation at Levels III and V, the 2-way variant is slightly slower per instance than the 1-way AVX-512 variant, while 4-way and 8-way batching recover substantial throughput gains. This illustrates that wider batching is beneficial only when the additional lane utilization outweighs packing, scheduling, and register-pressure overhead. To keep the baseline convention consistent with the other performance tables, all speedups are defined relative to the scalar C implementation.

Even without cross-instance batching, the single-instance AVX-512 implementations outperform the optimized ASM backend at every security level: by

Level	Impl.	#Inst.	Cycles	Cyc./Inst.	Speedup
I	C	1	1904	1904	1.00×
	ASM	1	828	828	2.30×
	AVX-512	1	646	646	2.95×
	AVX-512	2	1054	527	3.61×
	AVX-512	4	1530	383	4.97×
	AVX-512	8	2786	348	5.47×
III	C	1	3239	3239	1.00×
	ASM	1	2026	2026	1.60×
	AVX-512	1	1216	1216	2.66×
	AVX-512	2	1986	993	3.26×
	AVX-512	4	2980	745	4.35×
	AVX-512	8	5572	697	4.65×
V	C	1	4561	4561	1.00×
	ASM	1	2914	2914	1.57×
	AVX-512	1	1654	1654	2.76×
	AVX-512	2	2798	1399	3.26×
	AVX-512	4	4152	1038	4.39×
	AVX-512	8	7856	982	4.64×

Table 4. 4-isogeny evaluation throughput under varying parallelization degrees. Speedups are relative to the scalar C implementation.

Level	Impl.	#Inst.	Cycles	Cyc./Inst.	Speedup
I	C	1	2145	2145	1.00×
	ASM	1	938	938	2.29×
	AVX-512	1	654	654	3.28×
	AVX-512	2	1176	588	3.65×
	AVX-512	4	1894	474	4.53×
	AVX-512	8	3232	404	5.31×
III	C	1	3418	3418	1.00×
	ASM	1	2072	2072	1.65×
	AVX-512	1	1128	1128	3.03×
	AVX-512	2	2326	1163	2.94×
	AVX-512	4	3694	924	3.70×
	AVX-512	8	6244	781	4.38×
V	C	1	4838	4838	1.00×
	ASM	1	2890	2890	1.67×
	AVX-512	1	1492	1492	3.24×
	AVX-512	2	3238	1619	2.99×
	AVX-512	4	5058	1265	3.82×
	AVX-512	8	8750	1094	4.42×

Table 5. $(2, 2)$-isogeny evaluation throughput under varying parallelization degrees. Speedups are relative to the scalar C implementation.

1.28–1.76× for 4-isogeny evaluation and 1.43–1.94× for $(2,2)$-isogeny evaluation. At the largest batch size, 4-isogeny evaluation reaches 5.47×, 4.65×, and 4.64× the throughput of scalar C at Levels I, III, and V, respectively. The corresponding $(2,2)$-isogeny speedups are 5.31×, 4.38×, and 4.42×. These gains are larger than the single-instance AVX-512 speedups, confirming that cross-instance batching contributes materially beyond the acceleration provided by the vectorized field representation alone.

6.4 SQIsign End-to-End Performance

Table 7 reports end-to-end SQIsign performance. We include the reference C implementation, the optimized scalar ASM backend, our AVX-512IFMA implementation, and the AVX-512 implementation combined with Qlapoti [BCE+25]. The direct AVX-512 implementation already improves all three operations at every security level, while Qlapoti further reduces the ideal-to-isogeny cost in key generation and signing. Compared with the optimized ASM backend, AVX-512 improves key generation by 1.08–1.28×, signing by 1.07–1.25×, and verification by 1.46–1.84× across Levels I, III, and V.

Verification benefits most from vectorization: at Level I it improves from 12.54M cycles in C to 3.94M cycles with AVX-512, a 3.18× speedup. Key generation and signing additionally execute quaternion-side routines that remain scalar in our implementation. This difference follows from Amdahl's law: verification is dominated by the curve-side computation optimized in this work, whereas key generation and signing retain a substantial scalar quaternion component.

To quantify this effect, Table 6 separates the reference-C running time into the quaternion-side computation and the remaining computation. The remaining category includes finite-field, elliptic-curve, pairing, isogeny, and ancillary computations.

Level	Operation	Quaternion-side	Remaining
I	KeyGen	40.86%	59.14%
	Sign	42.92%	57.08%
III	KeyGen	41.64%	58.36%
	Sign	43.00%	57.00%
V	KeyGen	31.81%	68.19%
	Sign	35.26%	64.74%

Table 6. Coarse breakdown of the reference-C running time into quaternion-side and remaining computation. The remaining category intentionally includes all non-quaternion work.

The quaternion-side fraction is substantial: it accounts for 40.86% and 42.92% of Level-I key generation and signing, respectively, and remains between 31.81%

and 43.00% across the measured parameter sets. Since this portion is unchanged by our direct SIMD implementation, Amdahl's law places a hard upper bound on the speedup obtainable from vectorizing only the remaining computation. At Level I, even an infinitely fast non-quaternion part would limit key generation and signing to approximately 2.45× and 2.33×, respectively, compared with the measured 1.76× and 1.71×. This decomposition does not apply to Qlapoti, which changes the higher-level workload.

Level	Impl.	KeyGen		Sign		Verify	
		Cycle	Speedup	Cycle	Speedup	Cycle	Speedup
I	C	81.54M	1.00×	185.29M	1.00×	12.54M	1.00×
	ASM	50.05M	1.63×	116.15M	1.60×	5.77M	2.17×
	AVX-512	46.21M	1.76×	108.05M	1.71×	3.94M	3.18×
	AVX-512 (Qlapoti)	28.11M	2.90×	68.81M	2.69×	3.94M	3.18×
III	C	206.08M	1.00×	481.06M	1.00×	32.04M	1.00×
	ASM	152.80M	1.35×	349.64M	1.38×	20.39M	1.57×
	AVX-512	132.47M	1.56×	293.40M	1.64×	11.50M	2.79×
	AVX-512 (Qlapoti)	97.21M	2.12×	253.19M	1.90×	11.50M	2.79×
V	C	355.81M	1.00×	820.28M	1.00×	64.19M	1.00×
	ASM	251.91M	1.41×	582.24M	1.41×	39.75M	1.61×
	AVX-512	197.17M	1.80×	466.21M	1.76×	21.63M	2.97×
	AVX-512 (Qlapoti)	151.48M	2.35×	370.87M	2.21×	21.63M	2.97×

Table 7. Overall SQIsign performance.

6.5 Cross-Scheme Validation on CORAL

SQIsign is our primary optimization target, but many of the schedules above are expressed in terms of finite-field and higher-dimensional isogeny operations rather than signature-specific logic. We therefore use CORAL [BBRS26] as a second end-to-end test. CORAL evaluates a restricted isogeny group action using two-dimensional 2-isogenies and can be used to construct a post-quantum non-interactive key exchange. This gives a useful separation between the high-level cryptographic primitive and the low-level arithmetic: SQIsign and CORAL have different protocol structures, yet both make intensive use of higher-dimensional isogeny computations.

We ported the same AVX-512IFMA arithmetic backend and reused the applicable vectorized higher-dimensional routines from our SQIsign implementation. Table 8 reports key generation and shared-key computation on the same i7-11700F platform. The reference C implementation is the common baseline for

all parameter sets. The available CORAL code additionally provides a Broadwell backend for `lvl5`; we report it separately rather than using it as the baseline, because no corresponding ASM implementation is available for the other tested parameter sets.

Parameter	Impl.	KeyGen Mcycles	KeyGen Speedup	Shared-key Mcycles	Shared-key Speedup
`p_500`	C	42.73	1.00×	21.58	1.00×
	AVX-512	31.28	1.37×	10.21	2.11×
`p_1000`	C	205.51	1.00×	102.86	1.00×
	AVX-512	156.72	1.31×	53.61	1.92×
`p_2000`	C	1517.88	1.00×	711.57	1.00×
	AVX-512	1144.19	1.33×	339.54	2.10×
`p_4000`	C	13806.37	1.00×	5637.13	1.00×
	AVX-512	10774.25	1.28×	2637.42	2.14×
`lvl5`	C	44.57	1.00×	21.49	1.00×
	Broadwell	33.09	1.35×	14.28	1.50×
	AVX-512	31.74	1.40×	8.72	2.46×

Table 8. End-to-end CORAL performance. Speedups are relative to the reference C implementation. The `lvl5` parameter set is the only tested set for which the available code also provides a Broadwell backend.

The transfer is consistently beneficial despite the change in cryptographic primitive. Key generation improves by 1.28–1.40×, while shared-key computation improves by 1.92–2.46× over the reference C implementation. On `lvl5`, AVX-512 is also 1.04× faster than the Broadwell backend for key generation and 1.64× faster for shared-key computation. The larger gains in shared-key computation suggest that this operation spends a greater fraction of its execution time in the vectorized arithmetic. More importantly, the results show that the same vectorized building blocks provide end-to-end improvements in a cryptographic primitive with a different high-level structure.

7 Conclusions

This work studies SQIsign vectorization as an algorithmic scheduling problem rather than as an isolated finite-field optimization. The main loops of several SQIsign primitives are inherently sequential, but their arithmetic dependency graphs still contain useful fine-grained parallelism. By combining intra-primitive scheduling, cross-instance batching, and cross-operation fusion, we keep the curve-side computation in a vector-friendly radix-2^{51} representation and expose four- and eight-way arithmetic at the points where the algorithm can use it.

Our AVX-512IFMA realization provides substantial end-to-end gains across all three SQIsign security levels. At Level I, the direct implementation is 1.76×, 1.71×, and 3.18× faster than reference C for key generation, signing, and verification; combining it with Qlapoti raises the first two gains to 2.90× and 2.69×. The smaller acceleration of key generation and signing is explained by the remaining scalar quaternion-side computation, which is not present in verification. This also identifies the next optimization boundary: fixed-precision quaternion arithmetic such as that of Kim *et al.* [KLKL25] may allow the vectorized curve-side gains to contribute a larger fraction of total signing performance.

The CORAL case study further shows that these techniques are not specific to SQIsign. CORAL is not a signature scheme and has a different high-level control flow, yet reusing our AVX-512IFMA and higher-dimensional vectorization techniques yields 1.28–1.40× key-generation speedups and 1.92–2.46× shared-key speedups over reference C. More generally, our scheduling methodology identifies parallelism at the algorithmic level through intra-primitive scheduling, cross-instance batching, and operation fusion.

Future work includes combining this approach with fixed-precision quaternion arithmetic, evaluating the schedules on other wide-SIMD architectures, and studying newer formulas for higher-dimensional isogenies. More broadly, the results suggest that implementation work on isogeny-based cryptography should consider vectorizability at the level of complete arithmetic primitives, not only at the level of modular multiplication.

References

AAA+24. Marius A. Aardal, Gora Adj, Diego F. Aranha, Andrea Basso, Isaac Andrés Canales Martínez, Jorge Chávez-Saab, Maria Corte-Real Santos, Pierrick Dartois, Luca De Feo, Max Duparc, Jonathan Komada Eriksen, Tako Boris Fouotsa, Décio Luiz Gazzoni Filho, Basil Hess, David Kohel, Antonin Leroux, Patrick Longa, Luciano Maino, Michael Meyer, Kohei Nakagawa, Hiroshi Onuki, Lorenz Panny, Sikhar Patranabis, Christophe Petit, Giacomo Pope, Krijn Reijnders, Damien Robert, Francisco Rodríguez Henríquez, Sina Schaeffler, and Benjamin Wesolowski. SQIsign. Technical report, National Institute of Standards and Technology, 2024. available at `https://csrc.nist.gov/Projects/pqc-dig-sig/round-2-additional-signatures`.

BBRS26. Andrea Basso, Giacomo Borin, Ryan Rueger, and Sina Schaeffler. CORAL: Faster isogeny group action for post-quantum NIKE. Cryptology ePrint Archive, Paper 2026/896, 2026.

BCE+25. Giacomo Borin, Maria Corte-Real Santos, Jonathan Komada Eriksen, Riccardo Invernizzi, Marzio Mula, Sina Schaeffler, and Frederik Vercauteren. sfQlapoti: Simple and efficient translation of quaternion ideals to isogenies. In Goichiro Hanaoka and Bo-Yin Yang, editors, *ASIACRYPT 2025, Part IV*, volume 16248 of *LNCS*, pages 174–205. Springer, Singapore, December 2025.

BDD+24. Andrea Basso, Luca De Feo, Pierrick Dartois, Antonin Leroux, Luciano Maino, Giacomo Pope, Damien Robert, and Benjamin Wesolowski. SQIsign2D-West: The fast, the small, and the safer. Cryptology ePrint Archive, Report 2024/760, 2024.

Ber06. Daniel J. Bernstein. Differential addition chains. [Online], 2006. `https://cr.yp.to/ecdh/diffchain-20060219.pdf`.

CFGR22. Hao Cheng, Georgios Fotiadis, Johann Großschädl, and Peter Y. A. Ryan. Highly vectorized SIKE for AVX-512. *IACR TCHES*, 2022(2):41–68, 2022.

CLN16. Craig Costello, Patrick Longa, and Michael Naehrig. Efficient algorithms for supersingular isogeny Diffie-Hellman. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part I*, volume 9814 of *LNCS*, pages 572–601. Springer, Berlin, Heidelberg, August 2016.

CMO98. Henri Cohen, Atsuko Miyaji, and Takatoshi Ono. Efficient elliptic curve exponentiation using mixed coordinates. In Kazuo Ohta and Dingyi Pei, editors, *ASIACRYPT'98*, volume 1514 of *LNCS*, pages 51–65. Springer, Berlin, Heidelberg, October 1998.

CS09. Neil Costigan and Peter Schwabe. Fast elliptic-curve cryptography on the cell broadband engine. In Bart Preneel, editor, *AFRICACRYPT 09*, volume 5580 of *LNCS*, pages 368–385. Springer, Berlin, Heidelberg, June 2009.

Dup25. Max Duparc. Superglue: Fast formulae for (2,2)-gluing isogenies. In Goichiro Hanaoka and Bo-Yin Yang, editors, *ASIACRYPT 2025, Part IV*, volume 16248 of *LNCS*, pages 372–400. Springer, Singapore, December 2025.

FJWY26. Luca De Feo, Li-Jie Jian, Ting-Yuan Wang, and Bo-Yin Yang. SQISign on ARM. Cryptology ePrint Archive, Paper 2026/394, 2026.

FLOR18. Armando Faz-Hernández, Julio Cesar López-Hernández, Eduardo Ochoa-Jiménez, and Francisco Rodríguez-Henríquez. A faster software implementation of the supersingular isogeny diffie-hellman key exchange protocol. *IEEE Transactions on Computers*, 67(11):1622–1636, 2018.

HEY22. Hüseyin Hışıl, Berkan Eğrice, and Mert Yassı. Fast 4 way vectorized ladder for the complete set of montgomery curves. *International Journal of Information Security Science*, 11(2):12–24, 2022.

Kan97. Ernst Kani. The number of curves of genus two with elliptic differentials. *Journal für die reine und angewandte Mathematik*, 1997(485):93–122, 1997.

KLKL25. Won Kim, Jeonghwan Lee, Hyeonhak Kim, and Changmin Lee. SQIsign with fixed-precision integer arithmetic. Cryptology ePrint Archive, Report 2025/1649, 2025.

LR12. David Lubicz and Damien Robert. Computing isogenies between abelian varieties. *Compositio Mathematica*, 148(5):1483–1515, 2012.

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.

Mum83. David Mumford. *Tata Lectures on Theta II*, volume 43 of *Progress in Mathematics*. Birkhäuser, 1983.

NS22. Kaushik Nath and Palash Sarkar. Efficient 4-way vectorizations of the montgomery ladder. *IEEE Transactions on Computers*, 71(3):712–723, 2022.

PRR+25. Giacomo Pope, Krijn Reijnders, Damien Robert, Alessandro Sferlazza, and Benjamin Smith. Simpler and faster pairings from the montgomery ladder. Cryptology ePrint Archive, Report 2025/672, 2025.

Rob24. Damien Robert. Fast pairings via biextensions and cubical arithmetic. Cryptology ePrint Archive, Report 2024/517, 2024.

Sta08. Katherine E. Stange. *Elliptic nets and elliptic curves*. PhD thesis, Brown University, May 2008.