



Microarchitectural Benchmarking of Isogeny-Based Cryptography: Latency and Memory Profiling of the Modern SQISign Family

Dominic Fox

University College London

Department of Computer Science

MSc Information Security

Submission Date: 1 September, 2026

Declaration

I, Dominic Fox, confirm that the work presented in my thesis is my own. Where information has been derived from other sources, I confirm that this has been indicated in the thesis.

Various AI tools, including Claude and Gemini, were used to assist with grammar, document formatting, and general English writing. All changes made by the AI model were reviewed and approved by the author. All content of the paper was written by the authors themselves, and the authors retain full responsibility for the accuracy, originality, and integrity of the work.

Dominic Fox

London, United Kingdom

1 September, 2026

Abstract

As post-quantum cryptography transitions toward standardisation, the isogeny-based digital signature scheme SQISign offers exceptionally compact signatures at the cost of substantial computational overhead. While the scheme has evolved from its foundational one-dimensional formulation to two-dimensional (2D-West, 2D-East) and high-dimensional (SQISignHD) variants, direct comparisons between these paradigms under a uniform microarchitectural harness remain limited. This work presents a comprehensive systems-level evaluation of SQISign variants implemented on the AArch64 architecture. To achieve this, we introduce a custom Rust-based benchmarking orchestrator designed to rigorously profile bare-metal CPU cycle latency and stack memory consumption.

Our results reveal significant architectural trade-offs: while SQISignHD accelerates signing latency by over $70\times$ relative to NIST Round 1 and $3.2\times$ over NIST Round 2 while reducing peak stack consumption by 75%, it incurs a $15.7\times$ verification latency penalty relative to NIST Round 2. Ultimately, this analysis provides unified empirical metrics to guide scheme selection and implementation strategies for next-generation isogeny-based signatures.

Acknowledgements

I would like to thank my supervisors, Philipp Jovanovic (UCL) and Maria Corte-Real Santos (ENS de Lyon), for their support during this research. Their advice and guidance were invaluable.

Contents

1	Introduction	6
1.1	The Quantum Threat	6
1.2	The Evolution of SQISign	6
1.3	Problem Statement	7
1.4	Structure of Thesis	8
1.5	Accompanying Code	9
2	Contributions	9
3	Related Work	10
3.1	The Evolution of SQISign	10
3.2	Implementation Asymmetries	11
3.3	Architectural Asymmetries	12
3.4	Methodological Granularity	13
4	Background	14
4.1	Terminology	15
4.2	The SQISign Σ -Protocol	16
4.3	One-Dimensional vs High-Dimensional Paradigm	17
4.4	Taxonomy of the Evaluated Variants	18
4.4.1	SQISign NIST Round 1	18
4.4.2	SQISignHD (Dimensions 4 and 8)	19
4.4.3	SQISign2D-West (NIST Round 2)	19
4.4.4	SQISign2D-East	20
5	Implementation	20
5.1	System Architecture	20
5.2	Rust Orchestration	22
5.3	C-Level Instrumentation and Telemetry Extraction	22
5.3.1	Execution Modes	22
5.3.2	Hardware-Level Cycle Extraction on Apple Silicon	24
5.3.3	Cache Warmup and Memory Management	24
5.3.4	Zero-I/O Timing Loops	25
5.3.5	The Julia Environment (SQISign2D-East)	25
5.4	Stack Profiling and SQISignHD Verification Port	26
5.4.1	Memory and Stack Profiling Methodology	26
5.4.2	Porting SQISignHD Verification to C	27

5.5	Methodological Anomalies and Pipeline Validation	28
5.5.1	Upstream PRNG and Seed Parsing Anomalies	28
5.5.2	Validation of Telemetry Extraction	29
6	Evaluation	30
6.1	Experimental Setup	31
6.1.1	Hardware Environment	31
6.1.2	Compilation Parameters	31
6.1.3	Workload Characterisation	32
6.2	Latency Measurement Results	32
6.2.1	Asymptotic Scaling Across NIST Security Levels	36
6.2.2	Visualisations of the Distributions	38
6.3	Observations from the Data	40
6.3.1	Execution Variance	40
6.3.2	Timing Side-Channels	42
6.4	Memory Profiling	44
6.4.1	NIST Round 2 Memory Overhead	46
6.5	Comparative Literature Synthesis	46
6.5.1	Methodology Validation: 32-bit vs. 64-bit Stack Footprints . . .	46
6.5.2	x86 Assembly vs. AArch64 Reference Implementation	47
6.5.3	The Performance Penalty of Julia	47
6.6	Chapter Summary	48
7	Conclusion	51
7.1	Limitations	51
7.2	Future Work	52

1 Introduction

1.1 The Quantum Threat

The impending arrival of cryptographically relevant quantum computers (CRQCs) poses a fundamental threat to modern public-key cryptography [16]. Contemporary cryptographic standards rely on the hardness of integer factorisation and discrete logarithms [15, 14], which are solvable in polynomial time using Shor’s algorithm [17]. Given the rapid advancements in quantum hardware, industry leaders have issued urgent directives to migrate digital infrastructure to Post-Quantum Cryptography (PQC) standards [18]. While the migration of encryption algorithms is largely driven by the ”Harvest Now, Decrypt Later” threat model [19], the transition for digital signatures requires equal urgency. Should a CRQC be realised, adversaries could forge signatures to impersonate secure servers, sign malicious firmware, or compromise Public Key Infrastructure (PKI), necessitating immediate migration to quantum-secure standards.

In response to this threat, the National Institute of Standards and Technology (NIST) initiated a global standardisation effort [20], culminating in the selection of lattice-based schemes, such as ML-DSA [11], as primary signature standards. While these schemes offer fast computation, they present severe engineering trade-offs. Their public key and signature sizes span several kilobytes [11, Table 2], introducing critical transmission overhead that may be unacceptable in bandwidth-constrained systems like IoT devices or tactical networks in contested environments. Furthermore, the reliance on lattice-based cryptography for both key encapsulation mechanisms and signature standards introduces a systemic vulnerability; an algorithmic breakthrough in lattice reduction could compromise the newly standardised schemes. To mitigate this, NIST launched an additional call for signatures seeking candidates with diverse hard problems and reduced signature sizes [13].

1.2 The Evolution of SQISign

Within this landscape, isogeny-based cryptography occupies a vital architectural niche, yielding exceptionally compact keys and signatures. Early isogeny schemes suffered from impractical computational latency, often requiring minutes to execute [30, Table 2]. However, the introduction of SQISign (Short Quaternion-Isogeny Signature) drastically accelerated this process, achieving signing times of approximately two seconds while maintaining signatures of just a few hundred bytes [2].

To mitigate the remaining latency bottlenecks, researchers subsequently pioneered higher-dimensional isogeny mappings. By replacing difficult one-dimensional path-finding problems with direct geometric transformations in higher dimensions, these

architectural shifts have reduced signing latencies to under 40 ms, bringing isogeny signatures to the threshold of real-world viability. However, this rapid, isolated evolution has fractured the ecosystem into multiple competing variants, complicating standardisation efforts.

1.3 Problem Statement

Despite these theoretical advancements, evaluating the practical microarchitectural trade-offs between SQISign variants remains highly challenging. An optimised implementation of the ML-DSA lattice algorithm still achieves signing throughputs orders of magnitude faster than equivalent SQISign implementations [9, 25]. Furthermore, contemporary literature lacks a comprehensive, standardised cross-comparison of computational and memory complexity across the SQISign family. Because these novel schemes are developed in isolation, utilise disparate programming languages, and target diverse hardware architectures, the community currently lacks a unified performance comparison. Crucially, as these experimental variants are not yet integrated into standardised APIs, they cannot be evaluated by established automated benchmarking frameworks like PQC-LEO or SUPERCOP [26, 12].

The prevailing reliance on isolated benchmarking practices makes it difficult to determine whether reported algorithmic speedups are absolute mathematical improvements or merely environmental artifacts of unequal testing parameters. To address this benchmarking gap, this investigation engineers a bespoke evaluation framework targeted at the AArch64 architecture (Apple Silicon). By integrating the foundational baseline (hereafter referred to as NIST Round 1), the current standardisation candidate (NIST Round 2), and the broader family of experimental multi-dimensional variants (SQISign2D-West, SQISign2D-West-Heuristic, SQISign2D-East, and SQISignHD) into a singular, isolated test harness, this research provides the empirical telemetry required to objectively evaluate the modern SQISign ecosystem. (Note: Because this research focuses exclusively on this specific family of algorithms, the *SQISign* prefix is frequently omitted from variant names for brevity in subsequent chapters, e.g., 2D-West, HD). Specifically, this work addresses the following research questions:

- **Microarchitectural Latency:** How do the execution cycle counts of key generation, signature generation, and verification compare across the reference implementations of the SQISign family when executed in a unified AArch64 environment?
- **Resource Dynamics:** What are the peak stack allocations and physical resident set sizes required by each variant?

- **Preliminary Side-Channel Security:** Do the foundational SQISign implementations (NIST Round 1 and NIST Round 2) exhibit observable statistical key leakage when subjected to Test Vector Leakage Assessment (TVLA)?

SQIPrime is another theoretical 2D variant but it falls outside the scope of this investigation [7]. At the time of writing, SQIPrime is implemented exclusively within SageMath, a high-level language. Because SageMath executes within an interpreted environment with substantial overhead, benchmarking its CPU cycles and stack memory allocations against the compiled C codebases of the other SQISign variants would produce scientifically invalid telemetry. Consequently, this evaluation is strictly bounded to variants capable of generating native machine code for accurate microarchitectural profiling. Under this criteria, the 2D-East implementation is included despite being written in Julia. Unlike SageMath, Julia uses a Just-In-Time (JIT) compiler to generate optimised, native AArch64 instructions, allowing its telemetry to be meaningfully compared against the C baselines [47].

1.4 Structure of Thesis

The remainder of this work is organised as follows:

- Chapter 2: Defines the scope of the software artifacts, the AArch64 benchmarking framework, and the empirical telemetry produced in this work.
- Chapter 3: Reviews the literature on post-quantum signature schemes, the timeline of SQISign developments, and existing benchmarking efforts within the field.
- Chapter 4: Provides the foundational mathematics and cryptography necessary to understand the SQISign architecture, covering core primitives such as digital signatures and sigma protocols, alongside supersingular elliptic curves and isogenies.
- Chapter 5: Describes the architecture of the custom benchmarking harness and memory telemetry setup.
- Chapter 6: Presents the empirical data, comparing cycle latency distributions and memory footprints, accompanied by analysis of the core trade-offs.
- Chapter 7: Summarises the findings, provides practical recommendations for scheme selection, and outlines directions for future research in isogeny-based cryptographic implementations.

1.5 Accompanying Code

The software artifact developed for this dissertation, including the Rust benchmarking orchestrator and all modified cryptographic submodules, is publicly available on GitHub at <https://github.com/DominicFox/sqisign-test-harness>. Additionally, the repository contains the complete raw CSV telemetry datasets and additional visualisations.

2 Contributions

To accurately measure the performance trade-offs of modern SQISign variants, a primary deliverable of this work is the design and implementation of a rigorous, benchmarking test harness. The specific contributions of this work include:

- **Engineered C Telemetry Binaries:** Engineered bespoke telemetry binaries integrated directly into the source cryptographic implementations. These isolated environments safely initialise cryptographic parameters and randomness, execute primitives with designated CPU warm-up cycles to prime instruction caches, and serialise high-precision timing and state data directly to binary dumps.
- **Developed a Custom Stack Memory Profiler:** Developed a custom memory profiling framework to empirically measure stack consumption. This was achieved by allocating isolated virtual memory regions via `mmap`, deploying `PROT_NONE` guard pages via `mprotect` to safely trap overflows, and utilising stack-painting techniques. By spawning bounded threads within these regions, the framework accurately captures the exact high-water mark of memory usage during primitive execution.
- **Integrated SQISignHD Verification in C:** Incorporated the SQISignHD signature verification algorithm into the native C testing environment. Because the verification logic was previously available exclusively as a high-level SageMath reference, this facilitates what is, to the best of the author’s knowledge, the first microarchitectural benchmark of the scheme’s verification phase. While Large Language Model (LLM) orchestration was employed for the initial transpilation to meet timeline constraints, the resulting implementation was rigorously validated against a comprehensive suite of SageMath tests to guarantee strict semantic equivalence.
- **Architected a Rust Benchmarking Orchestrator:** Built a high-performance test orchestrator in Rust to automate the execution of the C-level telemetry binaries across continuous batches. To ensure the integrity of the execution latency

distributions, the orchestrator implements active thermal monitoring and enforces strict CPU cooldown heuristics, preventing thermal throttling from introducing hardware artifacts into the data.

- **Produced AArch64 Telemetry Dataset:** Designed a testing methodology to isolate sources of algorithmic jitter by capturing telemetry across three distinct operational paradigms: standalone key generation, fixed-key with random-message iterations, and fixed-message with random-key iterations. This suite was executed natively on Apple Silicon across all evaluated variants (NIST Round 1 [10], NIST Round 2 [9], 2D-West [1], 2D-West-Heuristic [1], 2D-East [6], and SQISignHD [45]) spanning NIST Security Levels 1, 3, and 5.
- **Constructed an Analytics Pipeline:** Developed a cohesive data pipeline that parses the raw C-binary telemetry into standardised CSV formats. This is paired with custom Python tooling to generate summary statistics and high-fidelity visualisations, actively exposing underlying performance bottlenecks and microarchitectural behaviours.

3 Related Work

3.1 The Evolution of SQISign

Since its foundational introduction in 2020, the SQISign protocol has evolved through rapid cycles of algorithmic optimisation. The original scheme established the Endomorphism Ring Problem as a viable foundation for compact post-quantum signatures but suffered from high computational latency due to its reliance on one-dimensional path-finding algorithms [2]. Subsequent refinements to the Deuring correspondence halved this latency, though the scheme remained significantly slower than lattice-based alternatives [3].

A pivotal architectural shift occurred following the cryptanalysis of the related SIDH key exchange in 2022 [4]. While the attack compromised SIDH, researchers leveraged its underlying mathematical techniques to revolutionise SQISign, using higher-dimensional isogenies. This led to the introduction of SQISignHD in 2023 [45], utilising four and eight-dimensional isogenies to achieve vastly improved signing speeds, albeit introducing a severe verification penalty. To balance this trade-off, 2024 saw the independent development of multiple two-dimensional variants: SQISign2D-West [1], SQISign2D-East [6], and SQIPrime [7]. These implementations successfully bypassed 1D path-finding bottlenecks without incurring the full verification overhead of higher dimensions. Consequently, the methodology underpinning SQISign2D-West was incorporated into the

scheme’s official NIST Round 2 specification [9], which subsequently advanced to Round 3 of the standardisation process, in May 2026 [31].

3.2 Implementation Asymmetries

A critical challenge in evaluating these rapid advancements is the software bias introduced by varying levels of implementation maturity. In isogeny-based cryptography, where execution is dominated by large-integer modular arithmetic, the choice of software backend fundamentally alters the relative performance ratios between competing variants. To systematically analyse the discrepancies in contemporary reporting, this work classifies cryptographic implementations into three operational tiers:

- **Tier 1 (Portable Reference):** Code written in high-level languages (e.g. generic C) without architecture-specific optimisations. This serves as the scientific baseline, isolating true mathematical efficacy.
- **Tier 2 (Intrinsic-Accelerated):** Code utilising architecture-specific intrinsics that map directly to specialised CPU instructions (e.g., AVX-512), introducing instruction-set bias [28, 8].
- **Tier 3 (Bare-Metal Assembly):** Hand-written assembly that manages registers and memory manually, representing peak engineering optimisation but masking underlying algorithmic complexity.

Applying this framework to the SQISign2D-West literature reveals methodological asymmetries [1]. The authors demonstrate a 15.8x mathematical speedup under strictly uniform Tier 1 conditions (Table 3). However, the headline “10x to 15x” speedup promoted in the broader literature appears to be derived from comparing a newly optimised Tier 2 implementation (utilising Intel Ice Lake intrinsics) against a legacy Tier 3 implementation optimised for an older Broadwell architecture (Table 4). This conflates mathematical innovation with hardware-generational acceleration.

Furthermore, cross-referencing Tier 1 and Tier 3 execution metrics reveals an optimisation saturation effect, a practical realisation of Amdahl’s Law [27]. As the core mathematical bottleneck is accelerated by higher-dimensional geometries, fixed overheads (e.g., memory management and control logic) occupy a disproportionately larger percentage of runtime. Consequently, older 1D variants exhibit higher relative gains from bare-metal optimisation than mathematically superior 2D variants (see Table 1). This justifies the necessity of the methodology employed in this research: orchestrating all variants under a unified CMake compilation target to prevent mismatched hardware optimisations from skewing empirical comparisons.

Algorithm Variant	Tier 1	Tier 3	Optimisation Factor
SQISign (NIST Round 1 [10], 2023)	4,781 M cycles	2,370 M cycles	2.02x
SQISign2D-West ([1], 2024)	290 M cycles	160 M cycles	1.81x
SQISign (NIST Round 2 [9], 2025)	163.1 M cycles	101.6 M cycles	1.61x

Table 1: Optimisation saturation analysis across the signing speeds of members of the SQISign family that offer both Tier 1 and Tier 3 code speeds (NIST Security Level 1).

3.3 Architectural Asymmetries

While the mathematical innovations of the 2D paradigm provide a theoretical path to efficiency, the practical utility of these gains in the current literature is heavily constrained by an x86_64 hardware bottleneck. The primary performance metrics for SQISign2D-West were derived on an Intel Xeon Gold 6338, leveraging architecture-specific, dedicated AVX-512 IFMA instructions [1, Table 4].

This heavy reliance on specialised Instruction Set Architectures (ISAs) creates a significant hardware-acceleration disparity when porting the algorithms to the resource-constrained or mobile environments (such as ARM Cortex-M) mandated by the NIST standardisation criteria [20]. Because isogeny algorithms require multi-precision arithmetic on exceptionally large integers (e.g., $p = 5 \cdot 2^{248} - 1$ [9]), efficient execution relies heavily on direct access to CPU carry flags. When architecture-specific backends (such as x86 intrinsics) are unavailable for a target platform, developers are forced to rely on portable software emulation. This emulation is evidenced by the fallback ADDC macros in the SQISign2D-West source code, which simulate hardware carry flags via constant-time bitwise logical operations. Such an approach inherently multiplies instruction counts and severely degrades performance, highlighting the penalty of executing unoptimised, portable code on non-x86 architectures.

```

1 // Digit addition with carry
2 #define ADDC(sumOut, carryOut, addend1, addend2, carryIn) { \
3     digit_t tempReg = (addend1) + (digit_t)(carryIn); \
4     (sumOut) = (addend2) + tempReg; \
5     (carryOut) = (is_digit_lesssthan_ct(tempReg, (digit_t)(carryIn)) | \
6                 is_digit_lesssthan_ct((sumOut), tempReg)); \
7 }

```

Listing 1: Portable ADDC Macro found in SQISign2D-West Source Code

This architectural sensitivity is empirically validated by comparing the theoretical SQISign2D-West paper against NIST Round 2 [9]. For the NIST submission, the

authors were mandated by standardisation guidelines to provide performance metrics for resource-constrained environments, specifically the ARM Cortex-M4 architecture. When stripped of x86 vectorisation advantages for standard evaluation, the theoretical 9 million cycles verification speed from 2D-West regressed to a cost of 123 million cycles on ARM Cortex-M4 (see [1, Table 4] and [9, Table 7], respectively). Consequently, addressing this literature gap requires actively shifting the benchmarking environment away from exclusively Intel architectures. By deploying a unified evaluation harness natively on an AArch64 architecture, this investigation provides diverse telemetry for consumer-grade hardware.

3.4 Methodological Granularity

The final limitation of the existing literature concerns the environmental scope and statistical granularity of performance reporting. In traditional cryptographic engineering, disabling hardware frequency scaling is the established gold standard for extracting deterministic cycle counts on x86 architectures. Consequently, the SQISign2D-West literature correctly reports metrics derived from this tightly controlled state. However, on modern ARM consumer platforms such as Apple Silicon, frequency scaling is dynamically enforced by the operating system based on thermal limits and cannot be artificially disabled. Because dynamic scaling is an unavoidable reality of modern consumer silicon, evaluating algorithms under active software thermal management provides a crucial, complementary metric to traditional static benchmarking.

Furthermore, previous empirical evaluations within the SQISign ecosystem have suffered from critically low sample sizes; for instance, NIST Round 1 derived its performance results from a median of merely 10 benchmark executions [10, Table 2]. While recent standardisation efforts demonstrate a commendable shift toward statistical rigour, with NIST Round 2 performance analyses evaluating implementations across diverse hardware ecosystems and deriving metrics from the median of 1,000 runs [9], methodological limitations remain. Reporting a single aggregated metric obscures the inherent algorithmic jitter of the heuristic search processes, making it impossible to diagnose whether execution variance is driven by key-induced or message-induced branching, and it fails to capture critical tail-end latencies.

Addressing these combined hardware and statistical deficits requires a comprehensive evaluation strategy. Rather than relying on artificially static hardware states and isolated median figures, this investigation establishes empirical transparency by extracting metrics across tens of thousands of iterations with forced thermal cooldowns. By evaluating execution distributions across both fixed-key and fixed-message paradigms, this work successfully isolates distinct sources of computational jitter. Coupled with

rigorous stack-memory profiling, this methodology moves beyond theoretical baselines to definitively map the practical architectural constraints and operational stability of the modern SQISign family.

4 Background

A digital signature scheme is a fundamental cryptographic primitive comprising three core algorithms: Key Generation, Signing, and Verification [37]. The generation phase outputs an asymmetric key pair; the private key enables a signer to compute a unique signature over a specific message, while the broadly distributed public key allows any third party to verify that signature’s validity. Together, these mechanisms guarantee the authenticity, integrity, and non-repudiation of digital communications. Without these cryptographic guarantees, digital infrastructure, including financial transactions, firmware updates, and legal contracts, would be fundamentally vulnerable to forgery and tampering. While classical digital signatures (such as RSA or ECDSA) rely on the computational hardness of integer factorisation or discrete logarithms [14, 15], these underlying problems are efficiently solvable by quantum cryptanalysis, via Shor’s algorithm [17]. Consequently, post-quantum alternatives must reconstruct these primitives upon entirely different mathematical foundations.

SQISign is a post-quantum signature scheme predicated on the computational hardness of the Endomorphism Ring Problem—the challenge of computing the complete set of structural mappings from a supersingular elliptic curve to itself [22]. On classical hardware, solving this problem requires $O(p^{1/2})$ operations, where p is the prime characteristic of the underlying finite field [38]. Because the numerical value of p grows exponentially relative to its bit-length n (i.e., $p \approx 2^n$), this effort is actually exponential in the size of the prime parameter, equating to $O(2^{n/2})$ operations. However, a quantum adversary can significantly accelerate this attack, reducing the complexity to $O(p^{1/4})$, or just $O(2^{n/4})$ operations [24, 1]. In practical terms, this quantum advantage dictates the required system parameters. To achieve a baseline security threshold of 2^{128} operations, a purely classical system would only require a 256-bit prime (since $2^{256/2} = 2^{128}$). To maintain that exact same 2^{128} threshold against a much faster quantum adversary, the underlying prime must be doubled to approximately 512 bits (since $2^{512/4} = 2^{128}$).

To standardise the evaluation of post-quantum algorithms, NIST defined five security levels [20, Section 4]. Rather than demanding precise bit-security claims, these levels are anchored to the computational difficulty of breaking established symmetric primitives under both classical and quantum attacks. The levels defined by NIST (hereafter referred to as Level 1, Level 3, and Level 5) are as follows:

Security Level	Definition
Level 1	At least as hard to break as a brute-force key search against AES-128.
Level 3	At least as hard to break as a brute-force key search against AES-192.
Level 5	At least as hard to break as a brute-force key search against AES-256.

Table 2: NIST Post-Quantum Security Levels evaluated in this work.

To contextualise the cryptographic variants evaluated in this work, it is first necessary to establish the mathematical primitives underlying the SQISign family and trace the protocol’s execution.

4.1 Terminology

For the purposes of cryptographic engineering, an elliptic curve defined over a field k (of characteristic different from 2 and 3, see [38, Chapter 2]) is most practically understood through its affine representation [39]. It is defined as the set of coordinate points (x, y) that satisfy the short Weierstrass equation:

$$y^2 = x^3 + ax + b$$

where $a, b \in k$ and the discriminant condition $4a^3 + 27b^2 \neq 0$ is satisfied to ensure the curve is smooth (meaning it has no sharp cusps or self-intersections) [33]. This set of coordinate points is appended with a special identity element known as the point at infinity, denoted as $\mathcal{O}$.

An isogeny $\phi : E_1 \rightarrow E_2$ is a non-constant rational map between two elliptic curves that preserves the group structure [33]. In practical terms this means points are mapped while maintaining consistent curve arithmetic (i.e. mapping two points and then adding them yields the same result as adding them first and then mapping the sum). An endomorphism is a specific classification of an isogeny where the domain and codomain are the identical curve, mapping $\varphi : E \rightarrow E$. The collection of all such endomorphisms for a given curve forms its endomorphism ring, $\text{End}(E)$, under the operations of addition and composition [33].

In the context of the SQISign protocol, the fundamental geometric objects are supersingular elliptic curves. This classification denotes a special family of elliptic curves that possess a highly complex algebraic structure, where the set of all endomorphisms of the curve is maximally large [32]. This rich internal structure is what allows the complex routing of the SQISign protocol to function. Furthermore, while standard elliptic

curves can operate over basic finite fields, SQISign requires defining these supersingular curves over a quadratic extension field, denoted as $\mathbb{F}_{p^2}$, where p is a large prime satisfying $p \equiv 3 \pmod{4}$. For formal definitions of supersingular endomorphism rings and quaternion algebras, see [33, 32].

The utility of isogenies to cryptography is better understood through graph theory. The supersingular elliptic curves and the isogenies between them form a massive, highly connected expander graph [52]. In this structure, the vertices represent the elliptic curves and the edges represent the isogenies connecting them. If ordinary elliptic curves were used instead, their isogeny graphs would form rigid, predictable networks vulnerable to simple traversal, making them unsuitable for cryptography. Instead, the highly connected supersingular graph provides the fundamental one-way asymmetry required for public-key cryptography. Traversing a known walk along this graph is a highly efficient, polynomial-time operation. However, the inverse problem of finding the path that connects a known starting curve to a distant destination curve is intractable for both classical and quantum computers.

The security of SQISign relies on a closely related variant of this path-finding asymmetry known as the Endomorphism Ring Problem: given a supersingular elliptic curve E , it is computationally intractable to compute $\text{End}(E)$ from scratch [22]. However, the viability of the entire signature scheme hinges on the fact that possessing this endomorphism ring provides a powerful trapdoor function, enabling the following polynomial-time operations [9]:

- Given E_1 , E_2 , $\text{End}(E_1)$, and $\text{End}(E_2)$, finding an isogeny $\phi : E_1 \rightarrow E_2$ reduces to a polynomial-time operation
- Given E_1 , E_2 , $\text{End}(E_1)$, and an isogeny $\phi : E_1 \rightarrow E_2$, finding $\text{End}(E_2)$ reduces to a polynomial-time operation

4.2 The SQISign Σ -Protocol

In cryptography, a Σ -protocol is a three-round interactive proof system between a prover and a verifier [36]. It structurally consists of three distinct phases: a commitment from the prover, a random challenge from the verifier, and a calculated response from the prover that relies on their secret knowledge. When processed through the Fiat-Shamir transform [34], this interactive identification scheme is converted into a non-interactive digital signature.

The asymmetric mathematical properties specified above form the basis of the SQISign identification protocol, which unfolds in the following phases. The entire system utilises a foundational supersingular curve, E_0 , where both the curve and its

endomorphism ring, $\text{End}(E_0)$, are pre-computed, static, public parameters:

1. **Key Generation:** The signer evaluates a secret, random isogeny walk $\tau : E_0 \rightarrow E_A$. The resulting destination curve, E_A , becomes the public key. The secret key is the exact path τ itself; because the origin ring $\text{End}(E_0)$ is public knowledge, possessing the secret walk τ allows the signer to mathematically derive the endomorphism ring of their public key, $\text{End}(E_A)$.
2. **Commitment Phase:** The signer generates a random commitment curve E_1 by evaluating another secret isogeny walk, $\psi : E_0 \rightarrow E_1$. Just as with the secret key, knowing the path ψ allows the signer to translate $\text{End}(E_0)$ into $\text{End}(E_1)$. The signer sends only the isolated destination curve E_1 to the verifier.
3. **Challenge Phase:** The verifier (or a cryptographic hash function via the Fiat-Shamir transform [34]) generates a random challenge isogeny $\varphi : E_A \rightarrow E_2$ from the public key to a new challenge curve E_2 .
4. **Response Phase:** Because the signer knows $\text{End}(E_A)$ (derived via their secret key τ), they can use the public challenge isogeny φ to compute the endomorphism ring of the challenge curve, $\text{End}(E_2)$. Now possessing both $\text{End}(E_1)$ (from the commitment) and $\text{End}(E_2)$, the signer leverages the polynomial-time property established previously to compute the required response isogeny $\sigma : E_1 \rightarrow E_2$.
5. **Verification:** The verifier confirms that σ is a valid isogeny of the required degree that successfully connects the published commitment curve E_1 to the challenge curve E_2 [1].

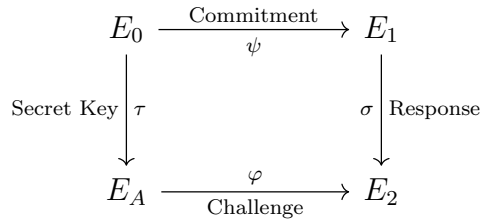


Figure 1: The general commutative diagram of the SQISign identification protocol. The evaluated variants primarily differ in how the response isogeny σ is computed.

4.3 One-Dimensional vs High-Dimensional Paradigm

While the Σ -protocol is theoretically sound, computing the response isogeny σ purely through geometric curve operations is computationally prohibitive. To resolve this,

SQISign leverages the Deuring correspondence [33, 21]. From a systems perspective, this mathematical theorem acts as translation between the intractable geometric problem of navigating between curves into a solvable algebraic problem within a quaternion algebra.

Instead of searching for a path geometrically, the signer maps the known endomorphism rings to algebraic structures and computes a connecting mathematical relationship (formally known as a left ideal). However, in the original one-dimensional version of SQISign, translating this algebraic solution back into a usable geometric isogeny path relies heavily on the KLPT algorithm [23].

KLPT fundamentally operates via a heuristic, probabilistic search. The algorithm must repeatedly attempt to solve complex equations until a mapping with specific structural properties (a smooth norm) is discovered. If an iteration fails, the algorithm is forced to discard its progress and restart the search with new randomised parameters.

To bypass the computational friction of the KLPT algorithm, recent variants (such as SQISign2D-West, 2D-East, and SQIPrime) lift the calculation into abelian varieties (higher dimensional generalisation of elliptic curves, see [40]), specifically product abelian surfaces of dimension 2, 4, or 8 [9].

By increasing the mathematical dimension, new geometric transformations (such as Kani’s Lemma [35]) become available. These higher-dimensional frameworks evaluate the response isogeny σ directly, completely bypassing the heuristic KLPT search required in one-dimension.

4.4 Taxonomy of the Evaluated Variants

While every SQISign variant conforms to the overarching Σ -protocol outlined in Figure 1, they diverge significantly in how the signer computes the response isogeny σ during the Response Phase. This single mathematical divergence is the root cause of the differing memory footprints and CPU cycle counts observed in the benchmarking phases of this research.

4.4.1 SQISign NIST Round 1

The original NIST Round 1 scheme operates exclusively on one-dimensional supersingular elliptic curves [2]. As previously established, computing the response isogeny σ through direct geometry is computationally infeasible in 1D. Instead, the implementation relies on the Deuring correspondence to translate the problem into an algebraic framework. The signer utilises the KLPT algorithm to probabilistically search until a suitable mathematical mapping (a smooth ideal) is discovered. Once found, this algebraic solution is translated back into the geometric response isogeny σ .

Expected System Impact: This 1D approach is mathematically designed to yield exceptionally compact signatures and efficient verification. However, the architectural reliance on the heuristic KLPT search is theoretically expected to severely bottleneck signature generation.

4.4.2 SQISignHD (Dimensions 4 and 8)

To eliminate the CPU bottlenecks inherent to the KLPT algorithm, the SQISignHD variant takes a fundamentally different approach by *embedding* the response isogeny σ into an isogeny of higher dimension (specifically 4 or 8) [45, 35]. Crucially, the signer does not compute these heavy higher-dimensional isogenies directly. Instead, the signer computes the images of a few specific points through σ . These points form a compact representation of the response and are transmitted to the verifier. The verifier then uses this data to evaluate the higher-dimensional mathematics, effectively offloading the geometric complexity away from the signer and avoiding the need for heuristic algebraic searches.

Expected System Impact: By completely eliminating the probabilistic search phase, this architecture is theoretically expected to substantially accelerate signature generation throughput. However, because much of the computational complexity is offloaded to the verifier, a corresponding increase in verification latency is anticipated.

4.4.3 SQISign2D-West (NIST Round 2)

Recognising the penalties incurred by dimensions 4 and 8, the SQISign2D-West variant, which forms the core architecture of NIST Round 2, identifies dimension 2 (abelian surfaces) as an optimal compromise between NIST Round 1 and SQISignHD [1, 9]. It retains the direct geometric evaluation inherent to the HD paradigm by embedding the response isogeny into a two-dimensional representation. Rather than defaulting to a full probabilistic search, the scheme introduces algorithms to directly translate algebraic quaternion ideals into these 2D geometric representations.

Expected System Impact: This architecture theoretically yields a highly balanced operational footprint. It is designed to capture the accelerated signature generation of the HD paradigm while ensuring that the computational complexity of verification remains sufficiently bounded. This prevents the stack memory consumption and verification latency from escalating to the extremes observed in higher dimensions.

The Heuristic Variant (2D-West-Heuristic): In addition to the strict mathematical framework, 2D-West includes a heuristic execution path. This version explicitly trades rigorous theoretical security proofs for heuristic assumptions to achieve faster signing speeds.

4.4.4 SQISign2D-East

Developed concurrently with the West variant, SQISign2D-East similarly operates on two-dimensional abelian surfaces to bypass the KLPT algorithm bottleneck [6]. The primary architectural divergence lies in the structural composition of the response isogeny. Whereas 2D-West is mathematically constrained to isogenies of a strictly smooth degree, 2D-East allows random isogenies of non-smooth degrees. This distinct algebraic approach allows 2D-East to evaluate the response without requiring the strict smoothness constraints of 2D-West.

Expected System Impact: While it shares the dimension 2 memory profile of the 2D-West variant, its differing approach to the response isogeny results in a distinct computational profile, offering alternative trade-offs between signature generation and verification CPU cycles.

5 Implementation

5.1 System Architecture

The primary objective of this implementation architecture is to empirically measure the CPU latency, and establish a framework for evaluating the stack memory footprint, of the SQISign variants discussed in Chapter 2.

To achieve this, a custom Rust repository, `\sqisign-test-harness`, was developed to act as the central orchestrator for the benchmarking environment. The system is deliberately decoupled into two distinct layers: independent C binaries for cryptographic execution and a parent Rust process for execution management and data aggregation.

The initial prototype of the benchmarking orchestrator was developed in Python to enable rapid iteration; however, the framework was subsequently migrated entirely to Rust. While Python is highly effective for high-level data manipulation, its runtime execution overhead and garbage collection unpredictability introduce unacceptable non-determinism when capturing cycle-exact microarchitectural telemetry. Rust was selected primarily for its zero-cost abstractions and deterministic memory management, which guarantee that the orchestrator’s execution lifecycle does not pollute the bare-metal CPU cycle counts of the cryptographic payloads.

Initially, an FFI (Foreign Function Interface) binding directly connecting the C cryptographic functions to the Rust runtime was considered. This approach was ultimately discarded due to the architectural complexity of reconciling Rust’s strict memory model with raw C pointers, alongside the need to resolve compilation type mismatches across the FFI boundary. Furthermore, FFI bindings introduce a risk of measurement over-

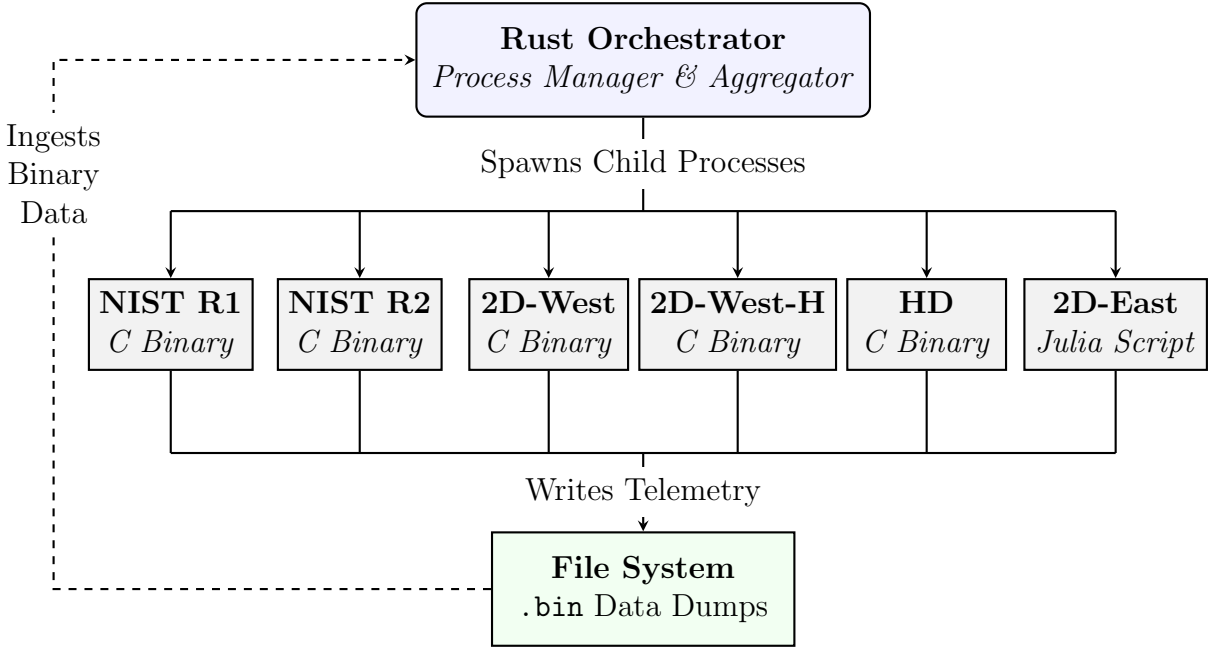


Figure 2: System topology of the `\sqisign-test-harness`. The Rust orchestrator maintains architectural parity by executing each variant as an isolated child process. NIST R1, NIST R2 and 2D-West-H are used as shorthand for NIST Round 1, NIST Round 2, and SQISign2D-West-Heuristic, respectively.

head that could obscure pristine CPU cycle counts. Conversely, because the upstream SQISign repositories natively support the compilation of independent benchmarking binaries, extending this existing build system to output custom telemetry binaries provided a robust mechanism to extract cycle counts with zero FFI-induced overhead, thereby ensuring maximum fidelity for performance profiling.

Alongside CPU cycle measurements, profiling the deterministic stack memory footprints is a critical requirement for evaluating post-quantum cryptographic implementations on embedded or constrained architectures. To achieve this without introducing microarchitectural noise, the stack profiling methodology relies on instrumentation embedded directly within the isolated C harnesses. By measuring the stack pointer boundaries at the invocation and conclusion of the cryptographic routines, the system captures a precise high-water mark of stack consumption.

To systematically aggregate the distinct SQISign implementations, a polyrepo architecture was adopted utilising Git submodules. This strategy maintained the provenance of the original, upstream codebases for the NIST Round 1 [41], NIST Round 2 [42], 2D-West [43], 2D-East [44], and HD variants [45].

Within each submodule, custom C binaries were injected to house the benchmarking logic. The native `CMakeLists.txt` configurations were subsequently modified to compile these specific files into standalone executable binaries tailored precisely to the

isolated testing environment.

5.2 Rust Orchestration

The Rust orchestrator governs the complete benchmarking lifecycle, encompassing process management, configuration switching, data ingress, and environmental control. By spawning the compiled C binaries as isolated child processes, the orchestrator guarantees a clean execution environment. Upon the successful completion of a subprocess, the parent process sequentially reads the generated binary dumps and parses the raw byte data back into a structured, CSV format for final analysis.

Given the high variance inherent in certain SQISign phases (such as the 1D KLPT search), mitigating hardware-induced jitter is critical. To address this, logic was integrated into the Rust orchestrator to monitor the host machine’s thermal state during execution loops. By interfacing directly with the operating system kernel, the thermal notification state is queried and recorded as an independent metric within the resulting CSV file, allowing for the systematic identification and exclusion of thermally throttled results.

Moreover, the orchestrator enforces a mandatory 15-second thermal cooldown period between execution batches. This allows the CPU to return to baseline temperatures and actively prevents thermal throttling from skewing the high-variance latency measurements. Ultimately, the orchestrator synthesises the thousands of raw data points from the binary dumps into standardised CSV formats for empirical evaluation.

5.3 C-Level Instrumentation and Telemetry Extraction

To extract precise CPU cycle counts without introducing observer effects or abstraction overhead, custom C binaries were injected directly into the upstream Git submodules. These files were designed to wrap the core cryptographic APIs (`crypto_sign_keypair`, `crypto_sign`, and `crypto_sign_open`) within strict hardware-level timing boundaries using the `cpucycles()` instruction.

5.3.1 Execution Modes

To ensure comprehensive benchmarking across different cryptographic threat models, the C harness was engineered with three distinct operational modes. These modes are dynamically controlled via command-line arguments passed by the Rust orchestrator:

- **Mode 0 (Key Generation Profile):** Isolates and profiles the creation of K distinct keypairs, serialising the resulting public and private keys to disk (`fixed_pk.bin` and `fixed_sk.bin`) for deterministic reuse in subsequent phases.

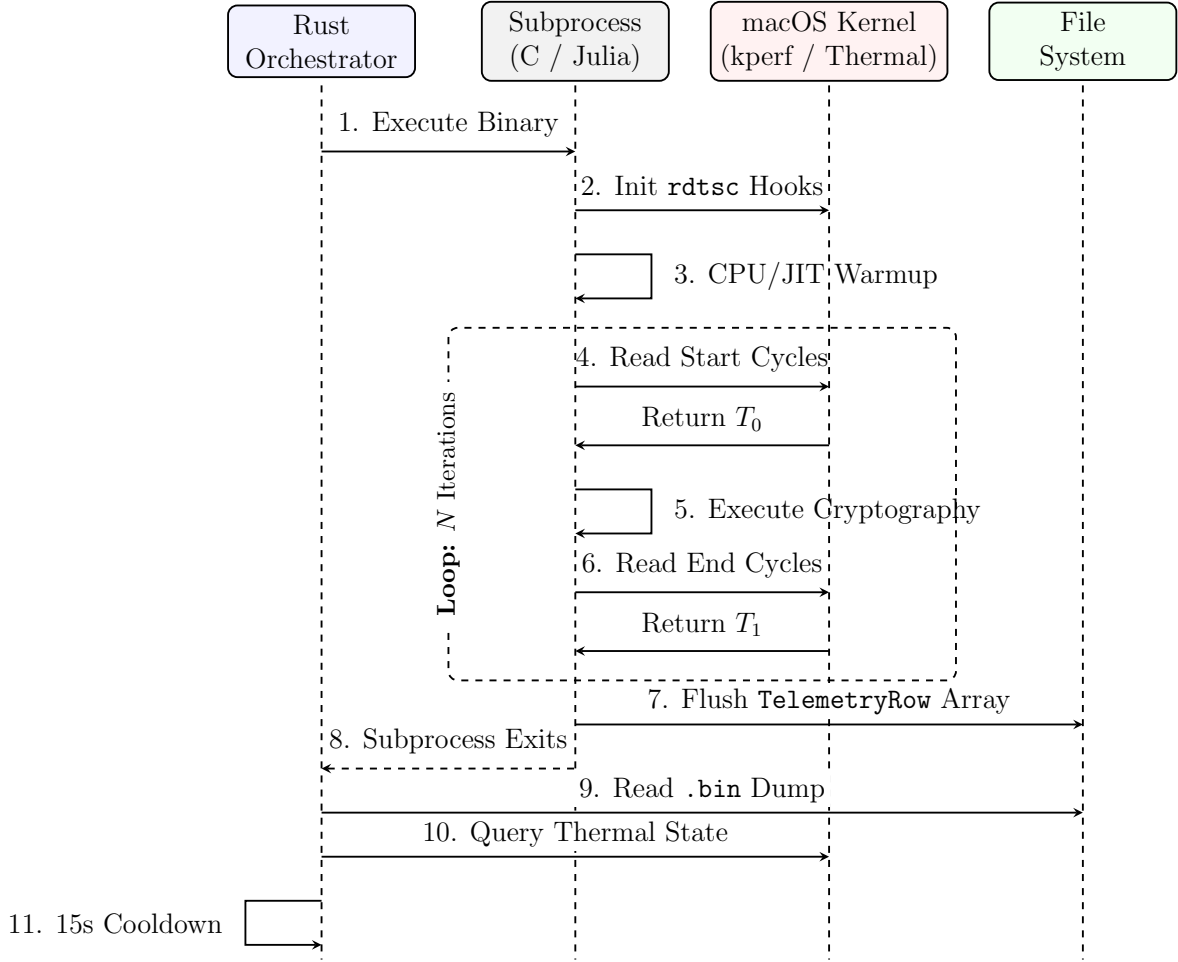


Figure 3: Sequence diagram detailing the execution lifecycle of a single benchmarking batch. The orchestrator enforces strict isolation, executing N iterations entirely in-memory to ensure hardware-level cycle extraction before applying thermal throttling mitigations.

- **Mode 1 (Fixed-Key, Random-Message):** Evaluates a single static keypair against M uniquely generated message payloads. This isolates the performance variance of the signing algorithm when the secret key path remains constant.
- **Mode 2 (Fixed-Message, Random-Key):** Generates a single static message challenge and evaluates it against K dynamically generated keypairs. This exposes the variance of key-dependent isogeny walks against a constant target.

5.3.2 Hardware-Level Cycle Extraction on Apple Silicon

Extracting CPU cycle counts presented a specific microarchitectural challenge. Standard x86 hardware natively provides the user-space `rdtsc` (Read Time-Stamp Counter) instruction for highly accurate and low-overhead timing telemetry [8]. However, the ARM-based architecture of the host Apple Silicon environment strictly limits direct user-space access to hardware Performance Monitor Counters (PMCs).

To circumvent this operating system restriction and maintain methodological parity with x86 benchmarking standards, a kernel-level hook was integrated into all C harnesses. Sourced from the NIST Round 2 codebase [42], this workaround dynamically loads macOS’s private and undocumented `kperf.framework` at runtime using `dlopen`. By mapping internal kernel functions (specifically `kpc_set_config` and `kpc_get_thread_counters`), the C harness configures the hardware PMCs to track raw core cycles (`CPMU_CORE_CYCLE`). The standard `cpucycles()` function is subsequently overridden to query these registers directly.

Methodologically, because interfacing with the private `kperf` API requires elevated kernel privileges, the benchmarking suite must be executed with `root` access. This ensures that the extracted telemetry reflects true bare-metal CPU cycles, entirely untainted by user-space timing inaccuracies, virtualisation layers, or operating system scheduling overhead.

5.3.3 Cache Warmup and Memory Management

A critical hardware consideration when benchmarking cryptographic algorithms is the state of the CPU cache. Cold-cache misses on the first few iterations of a loop artificially inflate cycle counts and skew statistical distributions. To mitigate this, the C harness implements a mandatory burn-in phase. Before any telemetry is recorded, the harness executes a configurable number of `warmup_runs`. These preliminary iterations execute the full mathematical operations, but the results and timing data are deliberately discarded, ensuring the CPU instruction cache and branch predictors are optimally primed.

Furthermore, to accommodate the shift from one-dimensional curves to the higher-dimensional abelian varieties present in the SQISign2D-West, 2D-East, and HD variants, explicit memory lifecycle management was required. While the NIST Round 1 implementation allows cryptographic keys and signatures to be managed as contiguous flat byte arrays, the complex mathematical structures of the subsequent variants possess significantly larger memory footprints. To prevent stack overflows during batch execution and to ensure that the deterministic stack profiling remains uncontaminated, the C harnesses strictly utilise variant-specific initialisation and finalisation subroutines (e.g., `public_key_init` and `public_key_finalize`). This guarantees that all large cryptographic buffers are allocated and freed exclusively on the heap.

5.3.4 Zero-I/O Timing Loops

To guarantee that the timing measurements reflect purely cryptographic execution, the harness completely decouples measurement from file I/O. During the active benchmarking loops, data is written exclusively to a pre-allocated array of `TelemetryRow` structs in volatile memory.

Only after the entire batch execution is complete does the harness perform file I/O operations, writing the array sequentially to a structured binary file (`sqisign_telemetry.bin`). The resulting telemetry rows are strictly formatted to include the key index, key generation cycles, signing cycles, and verification cycles. This architecture ensures that the operating system’s disk scheduling does not interrupt or artificially inflate the recorded CPU cycles.

5.3.5 The Julia Environment (SQISign2D-East)

While the majority of the evaluated variants were implemented in C, the upstream SQISign2D-East submission is authored in Julia [44, 47]. Benchmarking a high-level, garbage-collected language against bare-metal C implementations introduces significant methodological variables. To ensure parity across the dataset, a custom Julia measurement script was engineered to align strictly with the C-level telemetry architecture.

To accurately extract CPU cycle counts from the Julia runtime, a Foreign Function Interface (FFI) bridge was established to utilise the same native timing logic. The existing C header was adapted into a source file, `bench_macos.c`, and compiled into a native dynamic library using the Clang compiler. The compilation command strictly required the `-fPIC` flag to generate Position-Independent Code, while the `-O3` flag applied maximum-level compiler optimisations to minimise instruction overhead. The resulting `libbench_macos.dylib` binary allowed the Julia implementation to successfully hook into the native cycle-counting logic via `ccall`, guaranteeing that all cycle

counts are derived from the same bare-metal Performance Monitor Counters. Although bridging languages via an FFI inherently introduces a marginal execution penalty, we claim that the maximum-level optimisations and the sheer volume of cycles consumed by the underlying cryptographic operations render this overhead statistically negligible.

Furthermore, execution within the Julia runtime required mitigating two language-specific artifacts that would otherwise corrupt the CPU telemetry:

- **Just-In-Time (JIT) Compilation:** Because Julia utilises a JIT compiler, the initial invocation of any function incurs massive compilation overhead. The standard warmup loops were explicitly adapted to force the compilation of all underlying **Nemo** algebraic structures and cryptographic routines prior to any active measurement.
- **Garbage Collection (GC) Isolation:** Unpredictable memory reclamation sweeps by the Julia garbage collector can introduce severe artificial latency spikes during execution loops. To prevent this, the script manually triggers a full collection cycle (`GC.gc()`) immediately prior to the cryptographic operation, and subsequently suspends the collector entirely (`GC.enable(false)`) for the duration of the hardware timing window.

By systematically isolating the GC and standardising the hardware timer, the Rust orchestrator is able to ingest the resulting binary dumps with the assurance that the extracted metrics reflect purely cryptographic execution cycles structurally comparable to the C variants.

5.4 Stack Profiling and SQISignHD Verification Port

5.4.1 Memory and Stack Profiling Methodology

Alongside CPU cycle measurements, profiling the deterministic stack memory footprints is a critical requirement for evaluating post-quantum cryptographic implementations on constrained architectures. Measuring the stack on the primary execution thread is inherently contaminated by initialisation routines and process startup overhead. Consequently, a bespoke probing mechanism was engineered within the C submodules to construct a strictly controlled stack environment.

This isolation is achieved by allocating an anonymous memory region via the `mmap` system call, ensuring the pages exist in the virtual address space but do not consume physical memory until explicitly written to. To prevent the cryptographic primitives from exceeding the allocated bounds and corrupting adjacent memory, the lowest page is explicitly protected using `mprotect` with the `PROT_NONE` flag. This acts as a manual

guard page, a necessity given that assigning a specific stack via `pthread_attr_setstack` suppresses the default guard pages typically installed by the POSIX thread library. A dedicated thread is subsequently spawned within this region to execute the target primitive, after which it is joined, measured, and unmapped.

The orchestrator captures two primary memory metrics: `stack_bytes` and `resident_bytes`. To measure `stack_bytes`, the probe first determines the stack origin by reading the stack pointer (`sp`) from within the newly spawned thread. On the target Apple Silicon AArch64 architecture, this must be accomplished using inline assembly rather than compiler built-ins such as `__builtin_frame_address`. The built-in function returns the frame pointer, which points to the saved frame pointer and link register pair near the top of the current frame, positioned above `sp`. Painting memory from the frame pointer downwards would erroneously overwrite the probe’s own local variables.

Once the true `sp` is acquired, the entire memory region from its base up to the `sp` is populated with a predetermined 64-bit constant. Because the stack grows downwards, this region represents the exact memory space available for the callee to utilise. The cryptographic primitive is then invoked, pushing stack frames downwards and overwriting the constants in the process. Upon the primitive’s return, the system scans upwards from the base of the region to identify the first word that deviates from the magic constant. This address denotes the deepest point reached during execution. Subtracting this address from the original `sp` yields a precise high-water mark accurate to eight bytes. The likelihood of the primitive coincidentally writing the exact 64-bit constant at its maximum depth is one in 2^{64} , rendering this edge case statistically negligible.

To measure `resident_bytes`, the identical thread configuration is employed without the memory painting phase, as painting necessitates writing to every page and therefore artificially forces physical memory allocation. Instead, the `mincore` system call is utilised to determine the residency status of each page in the allocated region. By aggregating the number of resident pages and multiplying by the system page size, the orchestrator accurately quantifies the precise volume of physical memory genuinely accessed by the cryptographic routines.

5.4.2 Porting SQISignHD Verification to C

Within the SQISignHD repository, a complete C implementation for KeyGen and Signing was available [45], but the verification logic was provided exclusively as a high-level reference model in SageMath [46]. To achieve end-to-end, bare-metal hardware benchmarking of the HD variant, this verification routine required direct porting into the C testing environment.

Given the strict temporal constraints of the research timeline, manual porting of

the verification primitive was unfeasible. Consequently, the SageMath reference was ported to C using machine-assisted translation. The initial architectural action plan was generated using Anthropic’s Claude 5 Fable [48], while the subsequent code implementation was executed by Claude 5 Opus [49]. Crucially, this translation represents the sole instance of Claude-generated source code across all software artifacts produced for this investigation. To ensure the integrity of the newly implemented C verification primitive, a rigorous validation pipeline was established. The output of the C implementation was systematically cross-referenced against the original SageMath test vectors. By guaranteeing that the ported C function successfully verified signatures generated by the upstream codebase and properly rejected invalid cryptographic configurations, the primitive was certified for active benchmarking. This ensured that the SQISignHD variant could be evaluated for cycle latency and stack utilisation with strict methodological parity to the other schemes.

5.5 Methodological Anomalies and Pipeline Validation

5.5.1 Upstream PRNG and Seed Parsing Anomalies

During the validation of the testing environment, a critical anomaly was identified in the upstream SQISign reference implementation regarding its Pseudo-Random Number Generator (PRNG) handling [42]. To enforce deterministic execution across benchmark iterations, the orchestrator was designed to supply a fixed cryptographic seed. However, the implementation consistently failed to ingest this fixed key and silently defaulted to randomised key generation, a behaviour initially suspected to be linked to macOS environment incompatibilities.

Subsequent source-code analysis of the reference binary revealed a fundamental flaw in its hex seed parser. The parser was incorrectly evaluating the seed input as base-10 rather than base-16 hexadecimal, causing it to prematurely terminate execution upon encountering any alphabetical character. As a result, a standard hexadecimal seed input such as `0557cd436372f66` was improperly truncated at the first letter, parsing merely as the decimal value `0557` (equivalent to `0x22d` in hexadecimal).

As illustrated in Figure 4, the root cause of the non-determinism stems from the initial `sscanf` evaluation on line 3. The `%u` format specifier explicitly instructs the parser to extract an unsigned decimal integer. When supplied with a standard cryptographic hex string (e.g., `0557cd43...`), the parser successfully extracts the initial decimal characters (`0557`) but immediately halts upon encountering the first alphabetical character (`c`). Because it successfully parsed the leading integers, `sscanf` returns `1`, satisfying the conditional statement and returning a false success on line 4. This entirely bypasses the correct base-16 evaluation on line 6, silently mutating the cryptographic seed into

```

1 static int parse_seed(const char *arg, uint32_t *seed)
2 {
3     if (sscanf(arg,  --seed=%u , &seed[0]) == 1)
4         return 0;
5
6     if (sscanf(arg,  --seed={
7         0x% PRIx32 , 0x% PRIx32 , 0x% PRIx32 , 0x% PRIx32 ,
8         0x% PRIx32 , 0x% PRIx32 ,
9         0x% PRIx32 , 0x% PRIx32 , 0x% PRIx32 , 0x% PRIx32 ,
10        0x% PRIx32 , 0x% PRIx32 } ,
11        &seed[0], &seed[1], &seed[2], &seed[3], &seed[4], &seed[5],
12        &seed[6], &seed[7], &seed[8], &seed[9], &seed[10], &seed[11])
13        == 12)
14        return 0;
15
16    return 1;
17 }

```

Figure 4: The upstream `parse_seed` function from the SQISign reference implementation [42]. The reliance on the `%u` format specifier in the initial conditional block causes silent truncation of standard hexadecimal seeds.

a truncated decimal value.

5.5.2 Validation of Telemetry Extraction

Extracting the granular, per-iteration telemetry required for non-parametric statistical evaluation presents a microarchitectural challenge. An initial approach involved modifying the upstream reference implementations to output verbose cycle counts via standard output for downstream regex parsing. To validate the integrity of the custom ‘zero-I/O’ binary serialisation pipeline detailed previously, a comparative statistical analysis was conducted between this regex-parsed C binary and the modified silent binary.

Once the raw execution telemetry was aggregated into CSV format by the Rust orchestrator, a Pandas-based pipeline was utilised to extract and sanitise the data for formal validation. Initial diagnostic testing assessed the normality of the extracted cycle counts across a large sample size ($N = 5000$). Using a standard significance threshold of $\alpha = 0.05$, Shapiro-Wilk tests yielded negligible p -values ($p \ll 0.05$) across all three primitives (Keypair, Sign, and Verify). This mathematically proves that the execution data heavily deviates from a normal distribution, firmly necessitating the use of non-parametric statistical models.

Initially, it was hypothesised that iteration i in both pipelines processed an identical message and keypair. Under this assumption, a Wilcoxon Signed-Rank test would be the appropriate dependent statistical model. However, further validation revealed that

the aforementioned PRNG parsing flaw rendered each iteration entirely independent. A Spearman’s rank correlation coefficient was calculated to definitively prove the lack of ordinal association between the datasets, confirming that chronological pairing was an invalid assumption. Consequently, the Wilcoxon Signed-Rank test was discarded in favour of the independent Mann-Whitney U test.

Applying the Mann-Whitney U test to the raw arrays ($N = 5000$) revealed that the Keypair and Verify primitives maintained statistically identical distributions regardless of the parsing technique ($p = 0.282 > 0.05$ and $p = 0.445 > 0.05$, respectively). Conversely, the Sign primitive yielded a statistically significant difference ($p = 1.17 \times 10^{-29} \ll 0.05$), conclusively proving that the bespoke silent binary implementation recorded a lower overall cycle count.

Primitive	Shapiro-Wilk (Binary)	Shapiro-Wilk (Regex)	Mann-Whitney U (p)	Rank-Biserial (r)	Result
Keypair	1.62×10^{-51}	1.21×10^{-52}	0.282	0.0124	Statistically Identical
Sign	1.77×10^{-45}	2.50×10^{-40}	1.17×10^{-29}	0.1306	Statistically Distinct
Verify	7.02×10^{-93}	1.69×10^{-93}	0.445	0.0088	Statistically Identical

Table 3: Statistical validation of the telemetry pipeline ($N = 5000$), confirming non-normal distributions and evaluating the statistical divergence between parsing extraction methods.

Crucially, this performance delta highlights the microarchitectural dangers of granular data extraction. The original upstream reference implementations mitigate output overhead by computing and printing only summary statistics at the conclusion of a benchmark run. However, attempting to extract high-resolution data via standard output forces the compilation of massive string-formatting instructions into the executable. For a highly complex and long-running function like SQISign signature generation, altering the physical layout of the compiled machine code in this manner induces slight but consistent CPU instruction cache misses which can affect function execution time.

Therefore, by entirely decoupling measurement from standard output, the custom binary serialisation pipeline successfully captures the necessary high-resolution telemetry without corrupting the instruction cache. This yields a strictly more accurate representation of true bare-metal cryptographic latency, validating the pipeline for the comprehensive evaluations conducted in Chapter 6.

6 Evaluation

This section details and discusses the results produced by the engineering effort described in Chapter 5. We start by looking into the exact experimental setup and testing environment, before presenting tables of results for timing and stack usage data, followed by some insights and discussion on the results.

6.1 Experimental Setup

To guarantee the reproducibility and microarchitectural validity of the benchmarking data, strict environmental controls, compilation parameters, and execution workloads were defined.

6.1.1 Hardware Environment

All empirical evaluations were conducted on a 2023 Apple MacBook Pro equipped with an Apple M2 Pro System-on-Chip (SoC) and 16 GB of unified memory. The system operated on macOS Tahoe (version 26.5.2, Darwin Kernel 25.5.0).

To eliminate user-space scheduling jitter, the Rust orchestrator was engineered to escalate its own process priority to the maximum allowable `nice` value (-20). Furthermore, the execution environment was heavily sanitised: the host was connected to continuous AC power with sleep states disabled via the `caffeinate` daemon, wireless interfaces (Wi-Fi and Bluetooth) were disabled, and all non-essential user-space processes were terminated.

To mitigate thermal throttling artifacts inherent to modern passive or semi-passive cooling architectures, the orchestrator enforced a strict 15-second idle cooldown period between batch executions. To verify thermal compliance, the orchestrator hooked into the macOS kernel thermal notification subsystem, explicitly polling for CPU throttling flags and appending the thermal state to the telemetry CSV outputs for definitive verification.

6.1.2 Compilation Parameters

To ensure deterministic binary generation across the polyrepo architecture, the following compilation toolchains were locked:

- **C Orchestration:** Compiled using Apple Clang 21.0.0 (Target: `arm64-apple-darwin25.5.0`, POSIX thread model). The CMake build system (version 4.3.3) and GNU Make (version 3.81) were utilised with the flags `-DCMAKE_BUILD_TYPE=Release` and maximum optimisation `-O3`. Macro redefinition warnings were suppressed via `-Wno-macro-redefined`. All C-level cryptographic variants were dynamically linked against the GNU Multiple Precision Arithmetic Library (GMP) version 6.3.0.
- **Rust Orchestrator:** Compiled via `rustc` 1.93.1 and `cargo` 1.93.1 (LLVM backend version 21.1.8).
- **Julia Environment:** Executed using Julia 1.12.6 (Commit 15346901f00), linked against the 64-bit OpenBLAS library (`libopenblas64.dylib`).

6.1.3 Workload Characterisation

To accurately model both steady-state production environments and structural algorithmic variance, latency telemetry was extracted utilising the operational execution modes previously defined in Section 5.3.1.

For the final evaluation, Modes 0, 1, and 2 were each executed over $N = 10$ independent batches, with each batch consisting of $M = 1,000$ iterations. Crucially, every batch was preceded by 15 discarded warm-up iterations to populate the instruction and data caches, mitigating cold-start penalties.

Conversely, the empirical stack profiling methodology required a reduced iteration ceiling, as stack consumption is mathematically deterministic. Stack telemetry was extracted using two batches of 50 iterations, executing over a fixed message and keypair. The absolute maximum values for both the peak virtual stack footprint (`stack_bytes`) and the physical memory mapping (`resident_bytes`) were recorded for analysis.

6.2 Latency Measurement Results

The following tables present the empirical macro-level latency distributions, measured in absolute native CPU Megacycles, across the three primary cryptographic operations: Key Generation, Signature Generation, and Signature Verification. The metrics are derived from approximately 10,000 iterations of the Mode 1 testing paradigm (Fixed Key, Random Message Payload) executed on the Apple M2 Pro microarchitecture. Each table presents one of the three standard NIST Security Levels (Level 1, Level 3, and Level 5).

In contrast to symmetric cryptographic benchmarking which frequently normalises latency to Megacycles per byte, this evaluation measures absolute Megacycles per operation. Post-quantum signature schemes inherently process varying byte volumes depending on the output sizes of their specific internal hash functions. Because this data expansion is a fundamental characteristic of the algorithm, normalising per byte may artificially skew the performance metrics. Measuring absolute cycles ensures a precise reflection of end-to-end cryptographic latency.

Crucially, the standard deviation (σ) reported in these tables is computed across the entire distribution of sample data, explicitly including all execution outliers. In cryptographic engineering, worst-case execution boundaries are critical metrics. Preserving these outliers in the variance calculations ensures that algorithmic variants with severe latency spikes or highly variable execution paths are accurately represented, preventing artificial smoothing of the performance data.

NIST SECURITY LEVEL 1: Timing Data (Megacycles)						
Metric	NIST R1	NIST R2	2D-West	2D-West-H	2D-East	HD
Key Generation						
N (Samples)	9999	9997	10000	20000	10000	10000
Min	1495.148	47.315	73.736	71.023	389.531	71.110
1st %ile	1543.326	48.005	74.987	72.086	397.865	72.153
Median	1691.393	52.269	82.042	78.114	433.529	78.393
99th %ile	2038.448	65.439	101.621	97.554	522.302	98.456
Max	2518.897	81.499	237.389	117.580	573.432	116.274
Std Dev (σ)	103.982	3.737	5.997	5.474	33.552	5.622
Signature Generation						
N (Samples)	10000	10000	10000	5000	10000	10000
Min	2425.501	105.815	230.640	96.466	1097.842	32.278
1st %ile	2479.238	109.392	236.503	99.222	1126.851	33.289
Median	2670.506	119.654	251.012	110.078	1233.005	37.320
99th %ile	3492.554	141.849	278.644	135.152	1490.154	52.825
Max	4636.960	166.895	556.130	2755.250	1756.530	82.960
Std Dev (σ)	211.865	6.908	10.400	38.25	75.706	4.136
Signature Verification						
N (Samples)	10000	10000	10000	5000	10000	10000
Min	48.000	8.143	16.216	16.023	198.949	127.704
1st %ile	50.508	8.304	16.438	16.227	211.529	131.015
Median	57.195	8.468	16.512	16.443	250.111	133.168
99th %ile	67.776	8.551	16.682	16.548	307.232	133.962
Max	77.184	8.576	16.776	16.751	355.917	134.809
Std Dev (σ)	3.793	0.053	0.077	0.075	20.740	0.562

Table 4: Macro-level latency distributions (in Megacycles) across cryptographic operations for Level 1. NIST R1, NIST R2 and 2D-West-H are used as shorthand for NIST Round 1, NIST Round 2, and SQISign2D-West-Heuristic, respectively.

NIST SECURITY LEVEL 3: Timing Data (Megacycles)						
Metric	NIST R1	NIST R2	2D-West	2D-West-H	2D-East	HD
Key Generation						
N (Samples)	10000	10000	10000	20000	10000	10000
Min	8785.213	130.510	195.357	188.174	762.616	188.786
1st %ile	8885.189	133.413	198.809	191.024	776.851	191.043
Median	9117.009	145.537	218.109	207.456	844.847	207.570
99th %ile	9485.218	209.911	275.861	257.411	1025.613	259.079
Max	9799.500	573.366	18838.167	365.126	1146.623	303.482
Std Dev (σ)	130.619	15.212	189.686	14.259	65.932	14.296
Signature Generation						
N (Samples)	9998	9998	7000	10000	10000	10000
Min	15917.672	291.657	600.946	254.171	2097.043	84.433
1st %ile	16031.789	299.827	616.258	260.871	2214.089	86.861
Median	16524.117	331.376	653.034	288.174	2436.321	95.779
99th %ile	18495.960	432.169	735.254	354.947	3066.537	132.807
Max	22249.635	777.256	15692.327	9194.227	4013.378	183.304
Std Dev (σ)	520.140	27.092	192.009	95.834	174.847	9.821
Signature Verification						
N (Samples)	10000	10000	7000	10000	10000	10000
Min	205.804	23.214	44.109	43.780	398.930	355.375
1st %ile	220.540	23.619	44.646	44.283	420.101	374.007
Median	246.482	23.919	44.794	44.820	488.739	376.646
99th %ile	275.283	24.149	45.191	45.612	584.202	378.661
Max	299.309	24.234	45.548	45.794	646.390	380.244
Std Dev (σ)	11.706	0.119	0.203	0.257	38.325	0.991

Table 5: Macro-level latency distributions (in Megacycles) across cryptographic operations for Level 3. NIST R1, NIST R2 and 2D-West-H are used as shorthand for NIST Round 1, NIST Round 2, and SQISign2D-West-Heuristic, respectively.

NIST SECURITY LEVEL 5: Timing Data (Megacycles)						
Metric	NIST R1	NIST R2	2D-West	2D-West-H	2D-East	HD
Key Generation						
N (Samples)	11999	9995	10000	20000	20000	10000
Min	30417.277	225.802	386.834	376.495	1212.694	375.936
1st %ile	30675.169	228.748	392.656	380.287	1255.299	380.950
Median	31169.963	248.971	428.311	409.530	1368.366	409.825
99th %ile	32027.590	317.113	529.353	501.392	1658.413	507.068
Max	32989.382	507.644	5788.477	654.519	1862.807	688.756
Std Dev (σ)	288.466	20.013	108.065	26.338	101.838	26.975
Signature Generation						
N (Samples)	10000	9997	7000	8000	10000	10000
Min	54228.296	507.095	1251.888	501.541	3286.288	166.266
1st %ile	54602.675	518.791	1280.098	512.073	3404.087	170.540
Median	55658.189	577.320	1350.382	563.726	3886.232	186.719
99th %ile	59361.146	741.127	1483.534	684.689	5250.517	258.836
Max	64024.680	958.866	68759.790	6585.718	7826.045	359.188
Std Dev (σ)	989.741	45.934	717.631	90.211	389.859	19.002
Signature Verification						
N (Samples)	10000	10000	7000	8000	10000	10000
Min	550.233	47.641	89.081	88.491	638.422	798.050
1st %ile	595.760	48.129	89.779	89.145	664.013	804.404
Median	681.332	48.595	90.030	89.891	730.756	831.721
99th %ile	783.949	49.057	90.835	90.416	847.184	836.322
Max	930.226	49.414	91.835	91.020	1031.020	838.886
Std Dev (σ)	40.583	0.222	0.415	0.323	39.287	5.057

Table 6: Macro-level latency distributions (in Megacycles) across cryptographic operations for Level 5. NIST R1, NIST R2 and 2D-West-H are used as shorthand for NIST Round 1, NIST Round 2, and SQISign2D-West-Heuristic, respectively.

6.2.1 Asymptotic Scaling Across NIST Security Levels

An analysis of the median latencies reveals profound differences in how the underlying isogeny mathematics scale across the evaluated schemes. As the security level increases, necessitating larger prime fields and consequently larger finite field operations, the computational cost of the SQISign protocol grows at significantly different rates depending on the algorithmic variant.

The most striking observation is the massive algorithmic improvement between NIST Round 1 and the modern higher-dimensional variants. NIST Round 1 exhibits severe scaling penalties, taking 6.19 times longer to sign at Level 3 compared to Level 1, and 3.37 times longer to step from Level 3 to Level 5. In contrast, NIST Round 2 and HD variants scale far more gracefully. The HD variant’s signing execution scales by only 2.57 times between Level 1 and Level 3, and 1.95 times between Level 3 and Level 5.

While HD holds the absolute advantage in signing speed, NIST Round 2 demonstrates exceptional verification efficiency. At Level 5, NIST Round 2 verifies a signature in just 48.6 Megacycles, rendering it over 17 times faster than HD (831.7 Megacycles) and nearly 14 times faster than the legacy NIST Round 1 (681.3 Megacycles).

These scaling factors prove that the shift toward dimension-two isogeny representations in NIST Round 2 successfully flattened the computational curve for higher security parameters, making Level 3 and Level 5 viable for real-world cryptographic deployment.

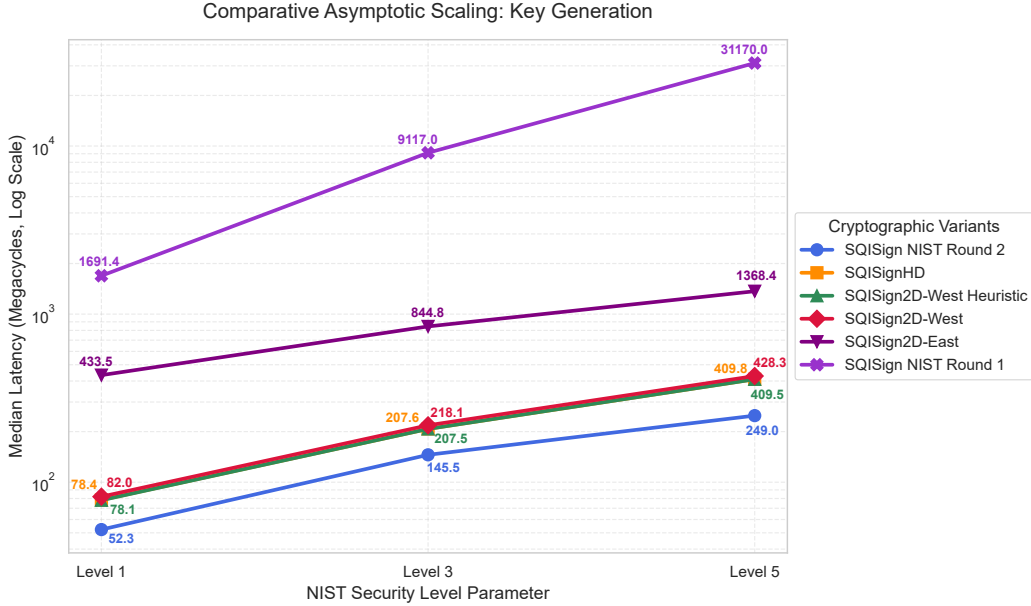


Figure 5: Comparative asymptotic scaling of key generation latency across SQISign variants, plotted by NIST Security Level.

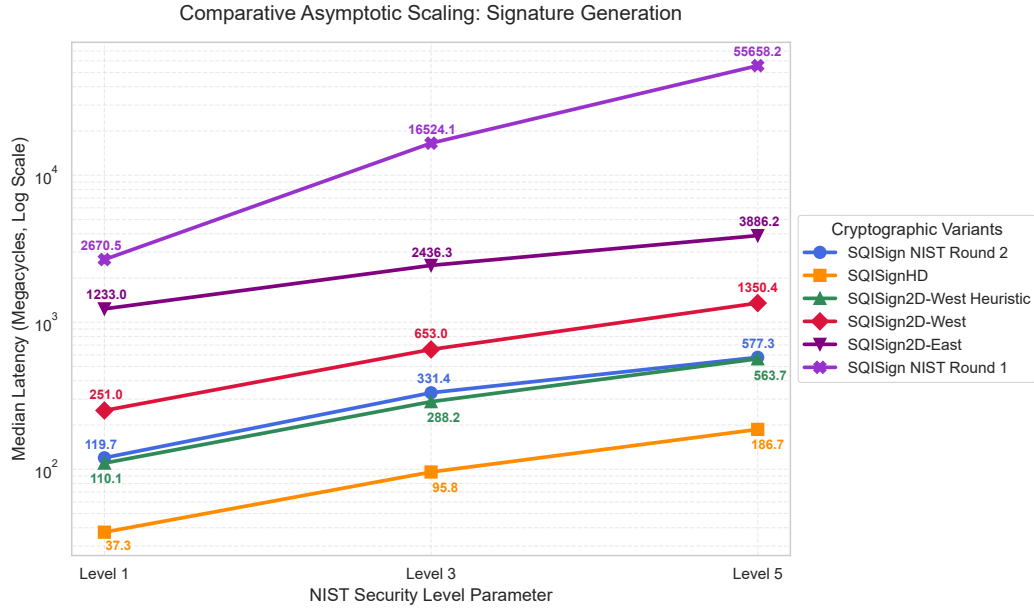


Figure 6: Comparative asymptotic scaling of signature generation latency across SQISign variants, plotted by NIST Security Level.

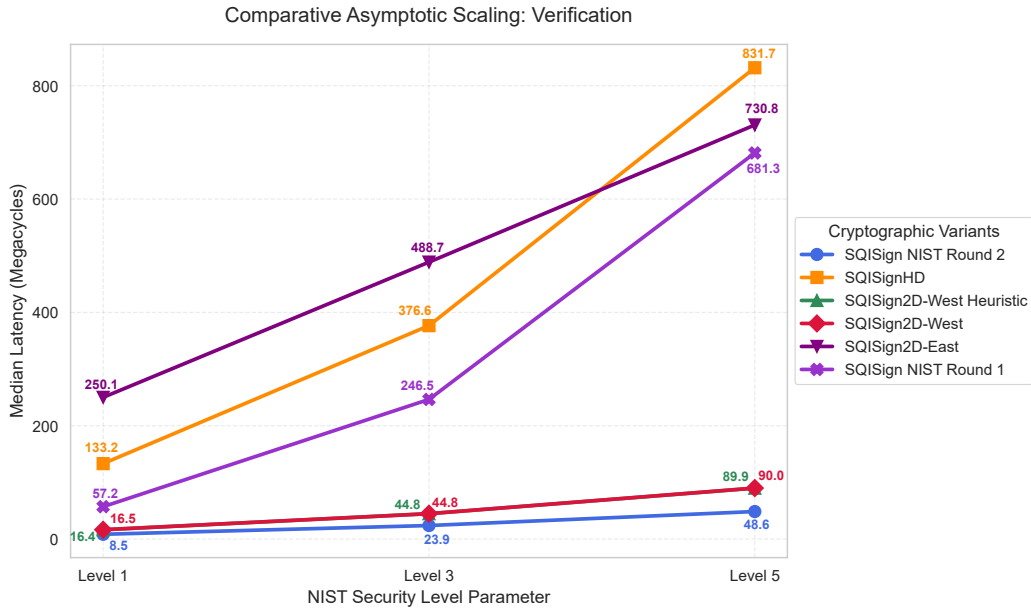


Figure 7: Comparative asymptotic scaling of signature verification latency across SQISign variants, plotted by NIST Security Level.

A comparative analysis across these NIST security levels demonstrates that no single SQISign variant is universally optimal. Instead, the appropriate selection is heavily dictated by deployment constraints. SQISignHD emerges as the most efficient algorithm for signature generation, signing a payload at Level 5 in 186.7 Megacycles, making it nearly three times faster than NIST Round 2 (577.3 Megacycles). Conversely, NIST Round 2 exhibits dominant performance in signature verification, completing the operation in merely 48.6 Megacycles. This highlights a critical architectural trade-off. A constrained Internet of Things (IoT) device broadcasting telemetry will inherently prioritise signature generation efficiency, making the HD variant the optimal choice. Conversely, a high-throughput web server responsible for terminating TLS handshakes will encounter a bottleneck at verification throughput, thereby favouring NIST Round 2.

The 2D-East variant is consistently the least performant scheme across all metrics. It is challenging to definitively attribute this latency to inherent algorithmic inefficiencies, suboptimal engineering within the reference repository, or the overarching execution overhead of the Julia environment. While the Just-In-Time compiled nature of Julia suggests it should theoretically achieve near-C performance, the aggregated data strongly implies that the current implementation suffers from a combination of the former two factors. Future work involving a native C port of 2D-East is strictly required to establish a direct, bare-metal comparison.

6.2.2 Visualisations of the Distributions

The resultant distribution graphs (Figures 8 to 10) vividly illustrate the heavy right-skew characteristic of the key generation and signature generation algorithms. This skew is directly attributable to the underlying rejection sampling loops inherent to isogeny walks. In the majority of iterations, the algorithm rapidly converges on a smooth path. However, probabilistic failures necessitate repeated sampling attempts, generating the observed right-tail latency outliers.

Conversely, the signature verification distributions are exceptionally tight, spanning a narrow band of approximately 8.2 to 8.6 Megacycles at Level 1. This deterministic profile is expected, as the verification protocol is a linear mathematical check devoid of probabilistic search loops. The presence of three distinct peaks within this narrow verification band highlights subtle runtime variance. While the narrow variance indicates a largely deterministic execution path, the emergence of secondary peaks suggests either minor conditional branch divergence within the verification logic or underlying microarchitectural artifacts.

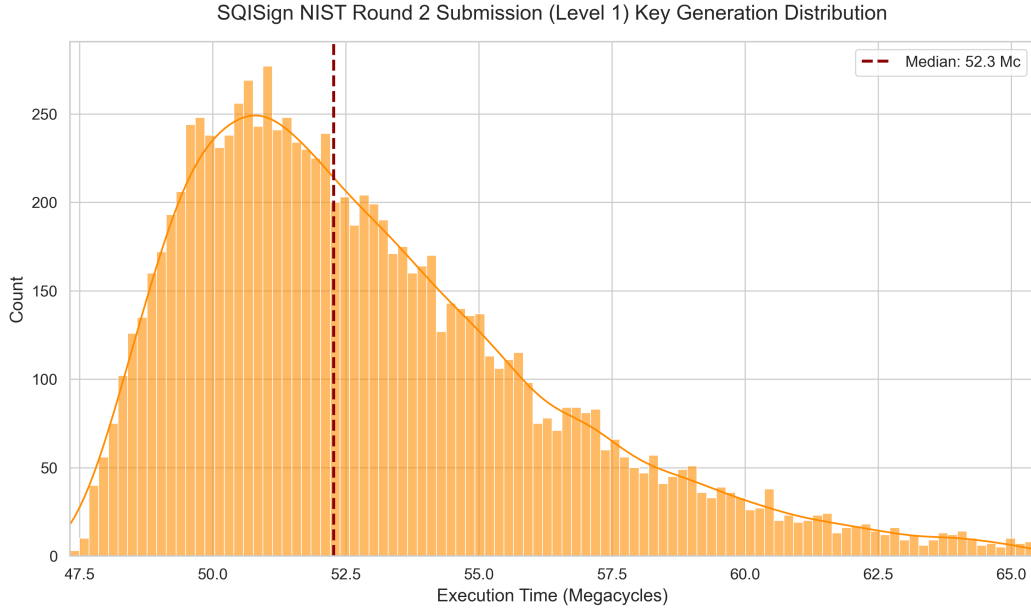


Figure 8: Latency distribution of NIST Round 2 Key Generation at Level 1. To preserve visual clarity, the x -axis is truncated at the 99th percentile, excluding extreme right-tail outliers caused by rejection sampling loops.

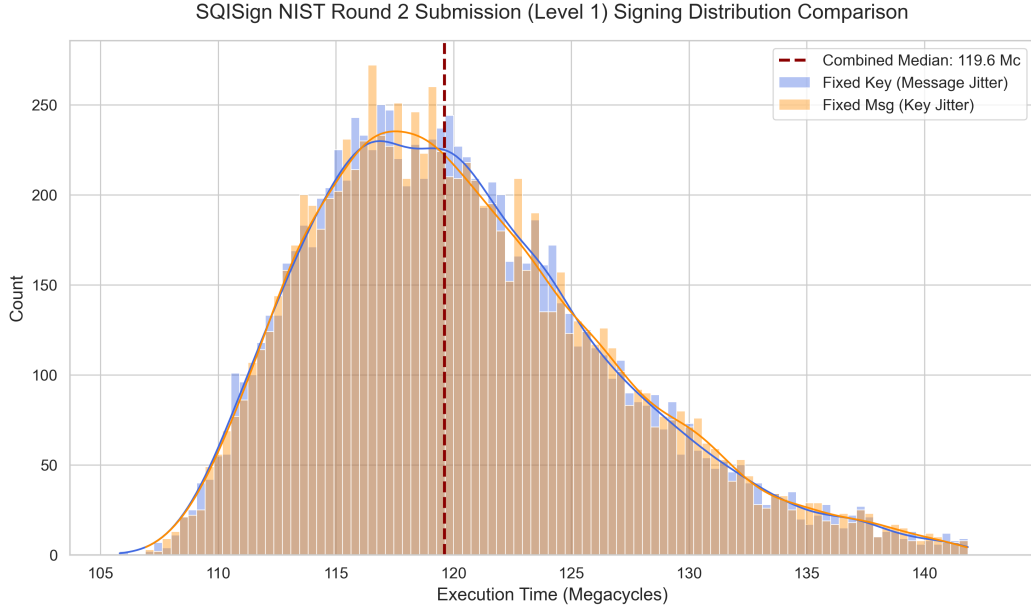


Figure 9: Latency distribution of NIST Round 2 Signature Generation at Level 1. The plot overlays Mode 1 and Mode 2 execution data to compare key-dependent and message-dependent results. The x -axis is truncated at the 99th percentile.

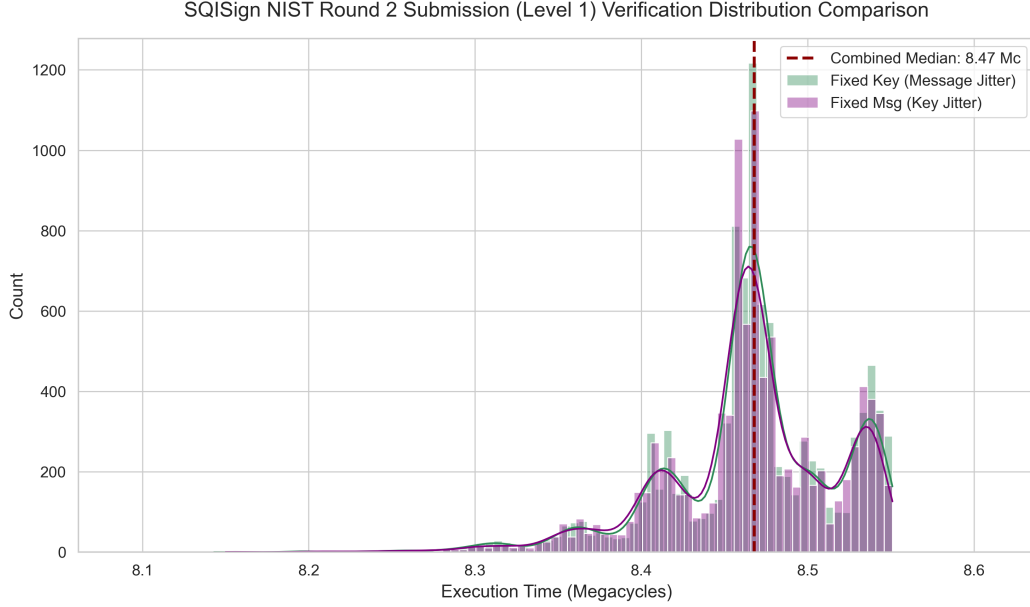


Figure 10: Latency distribution of NIST Round 2 Signature Verification at Level 1. Mode 1 and Mode 2 telemetry are overlaid to demonstrate structural consistency. The data is truncated at the 99th percentile.

A more complete set of visualisations is available in the accompanying GitHub repository: <https://github.com/DominicFox/sqisign-test-harness>

6.3 Observations from the Data

6.3.1 Execution Variance

While aggregated summary statistics illustrate general efficiency, evaluating algorithmic variance requires isolating the possible sources of execution jitter. Because SQISign signature generation relies on probabilistic isogeny path finding, runtime fluctuations can stem from two primary variables: the arbitrary message digest being signed, or the underlying geometric structure of the private key.

To isolate these factors, variance was evaluated in two statistical dimensions across both Mode 1 (1 Fixed Key, 1000 Random Messages) batches and Mode 2 (1 Fixed Message, 1000 Random Keys) batches:

- **Intra-Batch Variance ($\bar{\sigma}$):** Calculated as the arithmetic mean of the standard deviations across individual batches ($\bar{\sigma} = \frac{1}{N} \sum_{i=1}^N \sigma_i$). In Mode 1, this isolates intra-key jitter driven purely by varying message payloads. In Mode 2, it isolates intra-message variance driven by differing keys.
- **Cross-Batch Variance ($\sigma_{\tilde{x}}$):** Calculated as the standard deviation of the individual batch medians ($\sigma_{\tilde{x}} = \text{std}(\tilde{x}_1, \dots, \tilde{x}_N)$). In Mode 1, this measures whether

certain keys are inherently slower. In Mode 2, it evaluates whether specific message digests present persistent signing penalties.

The full range of results were intentionally retained in this analysis. As a result, the computed standard deviations include worst-case performance values. In isogeny-based signatures, extreme maximums can reflect rejection sampling loops within the underlying scheme. Truncating these data points would artificially conceal critical worst-case execution bounds.

Signature Generation Variance Analysis (Megacycles)							
Scheme	Metric	Level 1		Level 3		Level 5	
		Mode 1	Mode 2	Mode 1	Mode 2	Mode 1	Mode 2
NIST R1	Intra-Batch ($\bar{\sigma}$)	211.073	221.895	561.486	531.904	970.299	981.289
	Cross-Batch ($\sigma_{\bar{x}}$)	19.907	6.256	50.436	14.085	221.910	26.475
NIST R2	Intra-Batch ($\bar{\sigma}$)	6.909	6.761	27.460	33.399	47.017	47.505
	Cross-Batch ($\sigma_{\bar{x}}$)	0.226	0.345	0.694	0.784	1.384	0.885
2D-West	Intra-Batch ($\bar{\sigma}$)	10.282	9.310	124.891	266.745	311.124	605.768
	Cross-Batch ($\sigma_{\bar{x}}$)	0.430	0.400	0.633	1.114	2.152	6.265
2D-West-H	Intra-Batch ($\bar{\sigma}$)	27.210	25.447	49.056	148.393	78.203	66.767
	Cross-Batch ($\sigma_{\bar{x}}$)	0.215	0.294	0.684	0.385	1.358	1.004
2D-East	Intra-Batch ($\bar{\sigma}$)	74.513	73.363	173.729	175.492	388.854	371.069
	Cross-Batch ($\sigma_{\bar{x}}$)	14.396	9.775	18.614	20.564	30.768	29.785
HD	Intra-Batch ($\bar{\sigma}$)	4.132	4.247	9.815	10.114	18.972	18.838
	Cross-Batch ($\sigma_{\bar{x}}$)	0.197	0.151	0.353	0.203	0.876	0.474

Table 7: Comparative analysis of algorithmic variance during signature generation across all schemes and security levels. Intra-Batch variance ($\bar{\sigma}$) reflects the mean of individual batch standard deviations, while Cross-Batch variance ($\sigma_{\bar{x}}$) reflects the standard deviation of batch medians. NIST R1, NIST R2 and 2D-West-H are used as shorthand for NIST Round 1, NIST Round 2, and SQISign2D-West-Heuristic, respectively.

The variance metrics recorded in Table 7 reveal some behavioural distinctions across the schemes. Most notably, SQISign2D-West appears to exhibit input sensitivity at higher NIST Security Levels. At Level 3, the Intra-Batch variance in Mode 1 is 124.891 Megacycles, whereas Mode 2 expands to 266.745 Megacycles. At Level 5, this separation widens, with Mode 1 recording an Intra-Batch variance of 311.124 Megacycles compared to 605.768 Megacycles in Mode 2.

This disparity demonstrates that the standard 2D-West signing routine may be sensitive to differing private key structures. Certain keypairs introduce additional search complexity, forcing extended iteration. Incorporating heuristic search paths (2D-West-H) substantially reduces the overall magnitude of this execution jitter, however the data

reveals that key-dependent sensitivity is not fully eliminated at intermediate NIST Security Levels. At Level 3, Mode 2 Intra-Batch variance in 2D-West-H (148.393 Megacycles) remains over three times higher than Mode 1 (49.056 Megacycles). Parity between the operational modes is only fully achieved at Level 5, where Mode 1 (78.203 Megacycles) and Mode 2 (66.767 Megacycles) converge to comparable variance levels while maintaining substantially tighter bounds than the baseline 2D-West implementation.

In contrast, SQISignHD establishes the highest degree of operational determinism across the entire evaluation suite. Beyond providing the fastest raw signing latency, the HD variant demonstrates exceptionally tight variance bounds. At Level 5, its Intra-Batch variance remains constrained below 19 Megacycles in both operational modes, while its Cross-Batch variance remains under 0.9 Megacycles. This proves HD provides a highly predictable execution profile, making it desirable for systems with strict latency requirement where unlucky execution spikes cannot be afforded.

6.3.2 Timing Side-Channels

To evaluate the presence of microarchitectural timing side-channels, Test Vector Leakage Assessment (TVLA) matrices were constructed [50]. TVLA is a high-precision statistical methodology designed to detect structural differences between two datasets. Consequently, the threat model assumes an adversary with access to a low-noise, cycle-accurate local execution environment, such as local malware, capable of extracting high-resolution timing traces.

The TVLA pipeline applies Welch’s t -test to determine whether two sample sets (e.g., 1,000 signature generations from Key A vs. 1,000 from Key B) are drawn from identical underlying distributions. Standard cryptographic literature dictates that a threshold of $|t| > 4.5$ indicates statistically significant information leakage, with a confidence level exceeding 99.999% [50].

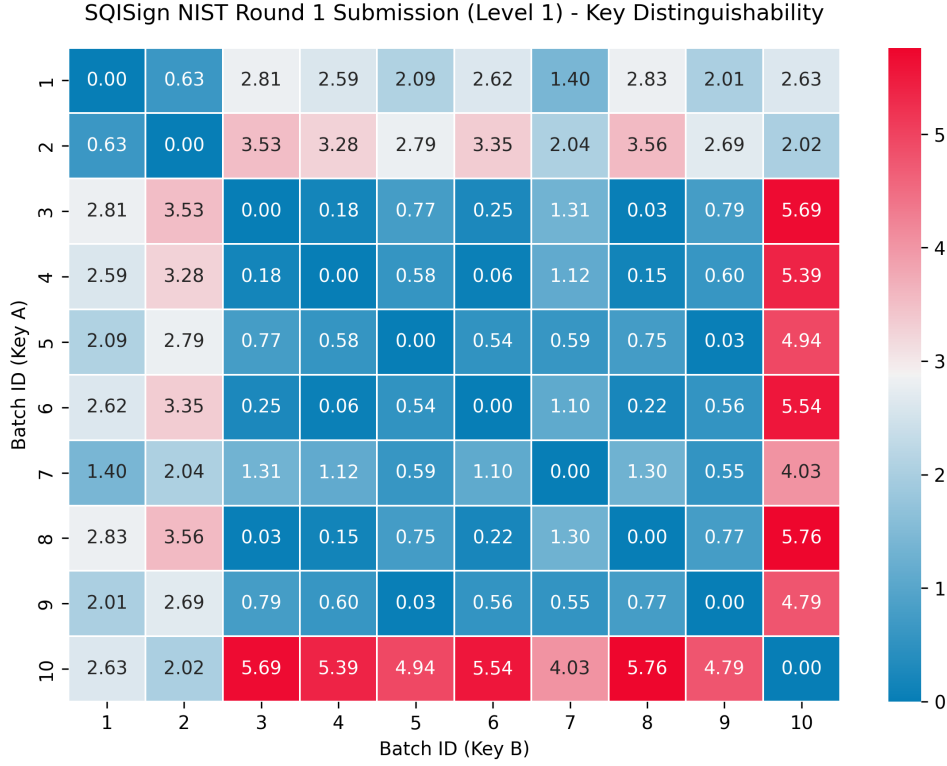


Figure 11: TVLA matrix for NIST Round 1 (Level 1), illustrating the statistical distinguishability of the key generated in Batch 10 against the other batches.

As demonstrated in Figure 11, the majority of keypairs are statistically indistinguishable; comparing Batch 3 to Batch 4 yields a t -value of merely 0.18. However, Batch 10 consistently exceeds the $|t| > 4.5$ leakage threshold against the baseline pool, peaking at a t -value of 5.76. This indicates that the specific mathematical structure generated for Key 10 naturally produces isogeny-walks that force the CPU down a slightly different execution path, which is highly detectable over 1,000 traces.

While a highly distinguishable key was absent in the initial ten-batch baseline for the more modern NIST Round 2 implementation, anomalous keys occasionally approached t -values of 2.0. To definitively assess whether NIST Round 2 remains susceptible to structural key leakage, a targeted search was executed across a broader dataset of 1,000 independent keys. To optimise search time, execution was restricted to 100 iterations per key. The ten keys exhibiting the highest variance were isolated and plotted in Figure 12.



Figure 12: Targeted TVLA matrix highlighting a statistically distinguishable key within the modern NIST Round 2 scheme, evaluated over a restricted sample size of 100 iterations.

Because Welch’s t -test scales directly with sample size, restricting the dataset to 100 iterations mathematically depresses the resulting t -values, making the 4.5 threshold significantly harder to breach [50]. Despite this constraint, a definitive statistical leak ($|t| > 4.5$) was still observed. Notably, the SQISign specification does not claim constant-time execution, therefore, rather than exposing a novel vulnerability, this data simply validates the authors’ theoretical assumptions [9].

6.4 Memory Profiling

This section evaluates the empirical memory footprints of the SQISign variants, a critical metric for deployment in resource-constrained environments. Table 6.2 details both the peak virtual stack consumption and the physical resident set size across all security parameters.

Notably, the 2D-East variant has been omitted from this memory analysis as the engineered harness works with C but not Julia, and creating a specific harness for this fell outside the scope of this evaluation.

The analysis evaluates two distinct metrics for stack usage. The virtual stack footprint (**stack_bytes**) records the absolute maximum depth reached by the stack pointer, directly answering the question: ‘*Will this algorithm execute within a strict hardware stack limit of X KB?*’ Conversely, the physical resident set size (**resident_bytes**) measures the exact volume of memory pages successfully allocated by the kernel, answering the question: ‘*How much physical RAM is actively consumed during execution?*’

Stack Data (Bytes)							
Scheme	Operation	Level 1		Level 3		Level 5	
		Stack	Resident	Stack	Resident	Stack	Resident
NIST R1	KeyGen	80816	81920	212064	212992	716960	720896
	Sign	85344	98304	216720	229376	720480	720896
	Verify	10304	16384	15264	16384	20384	32768
NIST R2	KeyGen	801200	802816	3448992	3457024	3029776	3031040
	Sign	808736	819200	3459088	3473408	3042512	3047424
	Verify	37120	49152	53296	65536	68712	81920
2D-West	KeyGen	356848	262144	779904	409600	1356632	622592
	Sign	374432	376832	804368	737280	1387976	1228800
	Verify	186000	114688	405408	147456	696552	163840
2D-West-H	KeyGen	356848	262144	779904	425984	1356632	622592
	Sign	361040	278528	784352	458752	1361336	688128
	Verify	183088	114688	400992	147456	690664	163840
HD	KeyGen	356848	262144	779904	442368	1356632	638976
	Sign	204896	114688	432064	131072	730968	147456
	Verify	163376	163840	242768	245760	322104	327680

Table 8: Empirical memory footprints across cryptographic primitives for all NIST Security Levels. Measurements reflect peak virtual stack bytes and physical resident set sizes. NIST R1, NIST R2 and 2D-West-H are used as shorthand for NIST Round 1, NIST Round 2, and SQISign2D-West-Heuristic, respectively.

A fundamental observation in the data is the frequent divergence between the virtual stack footprint and the physical resident bytes. For instance, during 2D-West Level 5 Verification, the algorithm reserves 696552 bytes of virtual stack space, yet consumes only 163840 bytes of physical RAM.

This discrepancy may indicate a sparsely utilised stack. When cryptographic C code declares massive, fixed-size arrays, the compiler shifts the stack pointer, reserving hundreds of kilobytes of virtual memory. However, the kernel only allocates physical RAM (in 16KB pages on Apple Silicon) when the execution path actively reads or writes to those addresses. Because verification is a relatively lightweight operation, the algorithm completes its execution having only touched a fraction of its reserved worst-case buffers, leaving the untouched virtual pages unmapped to physical memory.

6.4.1 NIST Round 2 Memory Overhead

The most striking observation within the memory telemetry is the massive stack requirement for NIST Round 2. While NIST Round 1 Key Generation at Level 3 requires merely 212 KB of stack space, the NIST Round 2 equivalent reaches over 3.4 MB.

This severe memory overhead appears to be an intentional architectural trade-off designed to meet strict NIST requirements, mandating specific acceptable thresholds for the algorithmic failure rate. Specifically, the failure rate of the `SuitableIdeals` subroutine is the cause of the observations. To achieve this reliability, the NIST Round 2 specification involves a re-randomisation setup utilising heavy precomputed data. This setup is detailed in Section 3.2.1.1 of the official SQISign specification [9], where the scheme defines an index of precomputed data setups denoted as $(O_t, J_t, E_t, (P_t, Q_t), (M_{t,1}, \dots, M_{t,4}))$.

Depending on the target security level, Section 4.2.2 of the specification dictates that a specific number of these setups (defined as *n_orders*, typically 6 or 7) must be stored. In the reference C implementation [42], this requirement results in exceptionally large arrays of precomputed structures, such as the `EXTREMAL_ORDERS` dataset located within `quaternion_data.c`. Because these massive static structures must be loaded and manipulated dynamically during key generation and signing, the compiler is forced to allocate unusually large stack frames, producing far greater virtual and physical memory footprints.

Furthermore, these re-randomisation parameters may explain the non-monotonic scaling observed between Level 3 and Level 5. While higher NIST Security Levels generally require larger prime fields, the specific tuning of the *n_orders* variable and its corresponding precomputed sets for the Level 3 prime structure ultimately forces a larger overall stack allocation (3.44 MB) than the Level 5 configuration (3.02 MB).

6.5 Comparative Literature Synthesis

To contextualise the empirical findings of this evaluation, the hardware-specific measurements were cross-referenced against the baseline performance metrics published in the official SQISign NIST Round 2 specification document [9].

6.5.1 Methodology Validation: 32-bit vs. 64-bit Stack Footprints

To validate the empirical accuracy of the custom AArch64 stack painting methodology, the verification footprints for NIST Round 2 were compared against the official submission metrics. The official documentation reports verification stack usage of 31 KB, 50 KB, and 64 KB for Levels 1, 3, and 5 respectively, evaluated on a 32-bit ARM Cortex-M4 microcontroller [9, Table 7].

Our orchestrator recorded 37.1 KB, 53.2 KB, and 68.7 KB on the Apple M2 Pro. This highlights around a 10% inflation which is likely an artifact of the transition from a 32-bit to a 64-bit architecture, where memory pointers and general-purpose registers double in width. As such, the alignment of these results with established literature strongly validates the accuracy of our measurement pipeline.

6.5.2 x86 Assembly vs. AArch64 Reference Implementation

An analysis of the official submission’s cycle counts reveals a noteworthy insight regarding modern cryptographic benchmarking [9]. The official documentation reports that a hand-optimised, assembly-level implementation targeting an Intel Broadwell or later architecture running on a desktop CPU (i7-13700K) achieves 43.3 Megacycles for NIST R2 Level 1 Key Generation. In contrast, the generic C reference implementation running on an Apple M3 Max achieves 48.8 Megacycles, while our Apple M2 Pro orchestrator recorded 52.3 Megacycles.

The fact that portable reference C code on an ARM-based laptop reaches near-parity with hand-optimised x86 assembly on a desktop processor is likely attributable to two core factors. First, the NIST reference implementation relies on the GNU Multiple Precision Arithmetic Library (GMP), which implicitly leverages highly optimised AArch64 assembly instructions for large-integer mathematics unless explicitly disabled [29]. Second, Apple M-series chips are architected to execute a high number of parallel instructions simultaneously per clock cycle [51]. This allows the hardware to process SQISign’s complex underlying mathematics with extreme efficiency directly from the standard compiled C code. Consequently, this demonstrates that the algorithm can achieve exceptional performance on modern hardware architectures without strictly requiring hand-written assembly optimisation.

6.5.3 The Performance Penalty of Julia

The inclusion of the 2D-East variant in this evaluation provided a direct mechanism to quantify the overhead of high-level numerical computing environments in cryptographic deployments. While the modern C-based variants exhibited highly competitive latencies, 2D-East consistently demonstrated severe performance degradation across all operations.

Crucially, this latency inflation should not be interpreted as a direct reflection of the underlying 2D-East mathematics. Because the 2D-East variant was evaluated using a Julia reference implementation, its execution profile fundamentally differs from the bare-metal C implementations of the other schemes. Even with strict experimental controls, such as execution warmup runs prior to measurement and aggressive Garbage

Collection suppression, the overhead of a higher-level language is unavoidable. Consequently, direct cycle-count comparisons between 2D-East and the C-based variants should be approached with caution, as the inflated execution times may largely represent language runtime artifacts rather than inherent cryptographic inefficiency.

6.6 Chapter Summary

The empirical evaluation of the SQISign variants reveals that the transition from the legacy NIST Round 1 specification to the modern dimension-two representations successfully established the scheme as a viable, high-performance digital signature standard. However, the data demonstrates that there is no single, universally superior SQISign variant. Instead, the cryptographic landscape is defined by a strict set of trade-offs across computational latency, algorithmic determinism, and memory footprint constraints.

To visualise these trade-offs, the peak virtual stack consumption and median execution latencies across all security levels were plotted for key generation, signature generation, and verification.

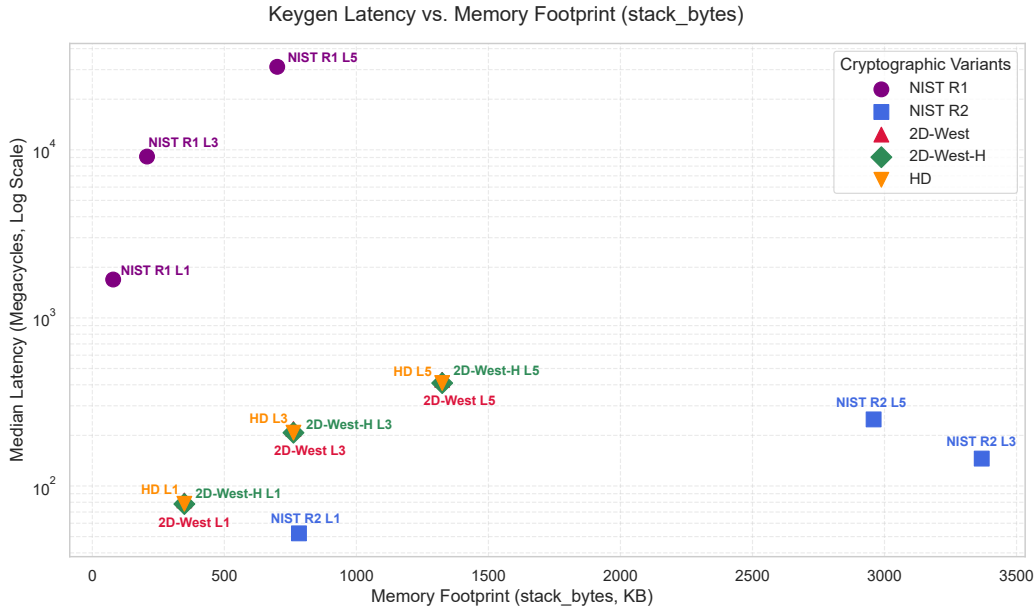


Figure 13: Median execution latency versus peak virtual stack consumption during SQISign key generation across all security levels.

NIST Round 2 is heavily penalised on the memory axis due to the massive static arrays required for its re-randomisation setup, pushing it far to the right. Conversely, HD and the 2D-West variants maintain much more conservative memory footprints, though their cycle counts scale aggressively at Level 5. NIST Round 1 achieves the

lowest stack footprint, at the cost of having the slowest execution time by an order of magnitude.

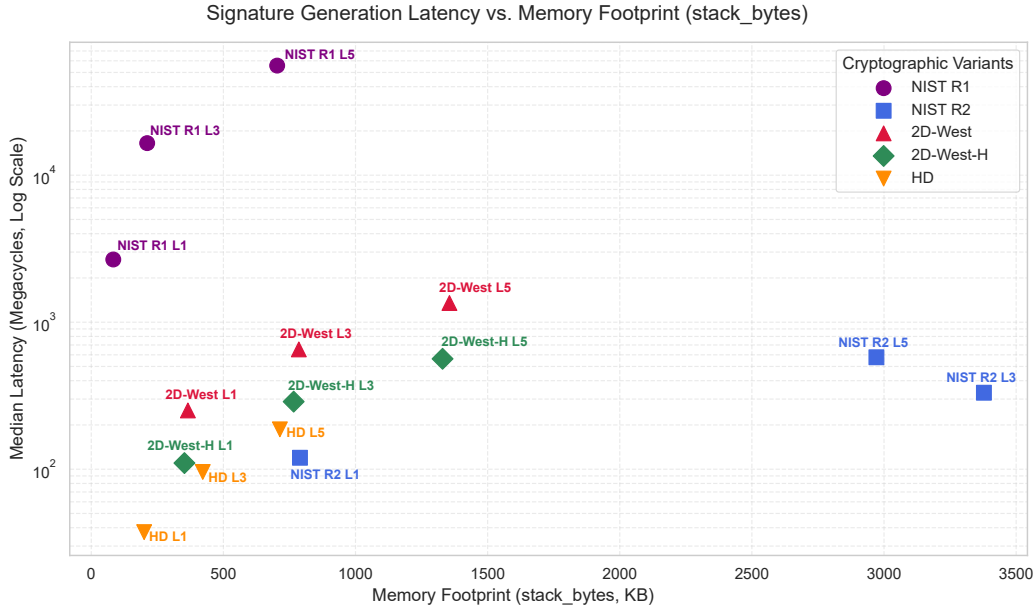


Figure 14: Median execution latency versus peak virtual stack consumption during SQISign signature generation across all NIST security levels.

For signing, there is a very clear winner. SQISignHD sits firmly at the optimal bottom-left edge of the modern variants. It achieves the lowest median latency across all NIST security levels while maintaining a stack depth comfortably under 1000 KB. The same trade-off between NIST Round 1 and NIST Round 2, as observed in KeyGen, exists here; NIST Round 1 maintains a very low memory footprint, while NIST Round 2 trades this for a fast signing speed.

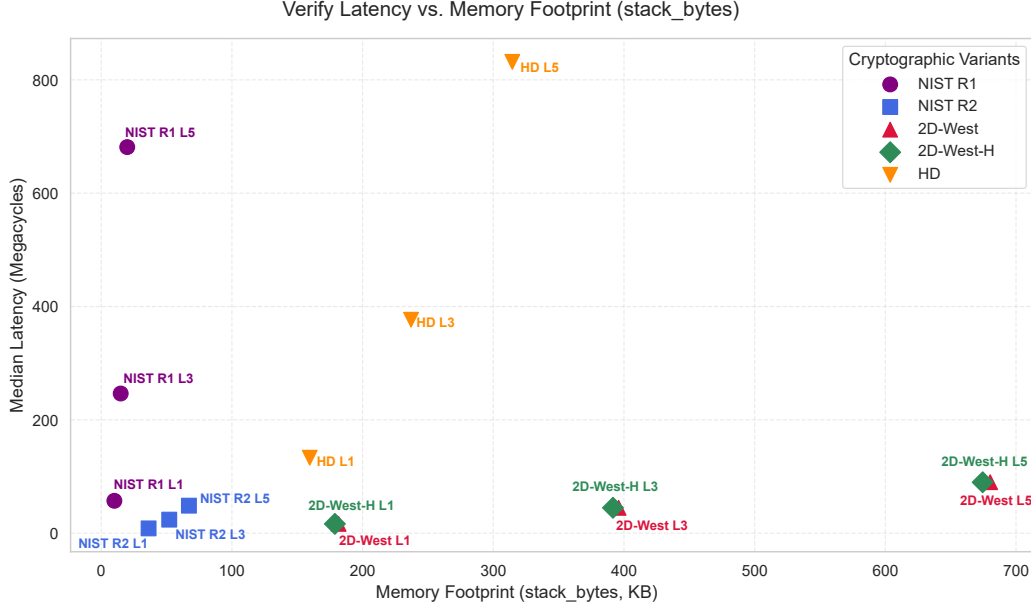


Figure 15: Median execution latency versus peak virtual stack consumption during SQISign verification across all NIST security levels.

Verification presents a completely inverted landscape. NIST Round 2 clusters tightly in the absolute bottom-left corner, achieving near-zero latency and a negligible stack footprint (under 100 KB). While HD dominated signing, it scales poorly for verification, with its Level 5 latency spiking to over 800 Megacycles.

This creates a clear contrast where HD is optimised for constrained provers, whereas NIST Round 2 is strictly optimised for constrained verifiers.

The core findings of this evaluation are summarised as follows:

- HD dominates signature generation, executing nearly $3\times$ faster than NIST Round 2 at Level 5 while consuming a fraction of the virtual stack memory. Conversely, NIST Round 2 remains the undisputed leader for signature verification, outperforming HD by a factor of $17\times$ at Level 5.
- The fast verification speeds and architectural stability of NIST Round 2 come at the cost of viability on constrained systems during key generation and signing. By statically allocating massive precomputed orders to reduce failure rates, NIST Round 2 demands over 3.4 MB of stack space at Level 3, potentially disqualifying its deployment on highly constrained microcontrollers.
- TVLA matrices confirmed statistical distinguishability ($|t| > 4.5$) under localised, cycle-accurate observation. This validates that the underlying algorithms are inherently variable and require further engineering effort to achieve constant-time execution.

Ultimately, the optimal choice of scheme requires architectural context. For constrained IoT hardware generating digital signatures, HD provides the most efficient balance of low stack allocation and rapid signing. For high-traffic TLS servers, the heavy prover stack penalty of NIST Round 2 is entirely justified by its unparalleled verification throughput.

7 Conclusion

This thesis evaluated the execution latencies, memory footprints, and side-channel resilience of the SQISign cryptographic signature scheme and its modern variants on the AArch64 architecture.

The ongoing NIST Post-Quantum Cryptography standardisation process has thus far heavily favoured lattice-based architectures, such as ML-DSA, due to their balanced computational profiles. However, as the cryptographic community transitions toward standardisation, the demand for protocols with minimal transmission overhead will become critical. The empirical baselines established in this work demonstrate that while isogeny-based cryptography remains computationally demanding, the modern SQISign variants have matured into highly viable, high-performance alternatives.

Rather than a single monolithic solution, the landscape is now defined by architectural trade-offs. While these schemes are highly effective when deployed in context-aware environments—matching specific variants to either constrained provers or high-traffic verifiers—the systemic vulnerabilities highlighted by the TVLA metrics decisively prove that the underlying algorithms remain inherently variable. Before these schemes can achieve formal standardisation, significant engineering effort is required to mask key-dependent isogeny walks and achieve constant-time execution.

7.1 Limitations

While the empirical methodology provides high-resolution telemetry, certain environmental boundaries constrain the broader applicability of the findings. First, the evaluation was orchestrated under the Darwin kernel. The resulting memory metrics, specifically the demand paging behaviour and physical resident set sizes, are fundamentally tied to the presence of a sophisticated Memory Management Unit (MMU). These specific memory optimisations do not extrapolate perfectly to bare-metal embedded targets, such as standard ARM Cortex-M4 microcontrollers, which lack dynamic page allocation and require contiguous physical RAM.

Second, the implementation of the SQISignHD verification algorithm utilised in this pipeline was a bespoke port authored specifically for this research. Unlike the mature

NIST reference codebases, this custom implementation has not yet been subjected to rigorous algorithmic optimisation or peer review by the wider cryptographic community. Therefore, a formalised, community-verified reference implementation could yield different execution bounds in future evaluations.

7.2 Future Work

While this evaluation establishes a comprehensive performance baseline on Apple Silicon, several avenues remain open for future research:

- **Porting 2D-East:** The severe latency penalties incurred by the Julia runtime emphasise the need for a pure C or Rust implementation of the 2D-East scheme to accurately assess its hardware-level viability.
- **Side-Channel Mitigation:** Given the proven statistical leakage under local observation, future work must explore constant-time algorithms, to mask key-dependent isogeny walks.
- **Stack Optimisation:** Investigating memory allocation strategies for NIST Round 2 could potentially reduce its massive 3.4 MB stack requirement, expanding its viability into embedded ecosystems.
- **Other Architectures:** While this study provides high-fidelity AArch64 telemetry, orchestrating an equivalent pipeline for x86-64 environments is crucially important to understand how these variants scale across differing architectures.

As the cryptographic community anticipates the forthcoming NIST Round 3 submissions, which will undoubtedly introduce further parameter tuning and algorithmic refinements, the findings presented here establish a definitive comparative baseline. By rigorously mapping the architectural trade-offs between stack consumption, execution latency, and microarchitectural distinguishability, this research provides the empirical foundations necessary to guide future optimisation efforts, ultimately advancing the formal standardisation of SQISign.

References

- [1] A. Basso, L. De Feo, P. Dartois, A. Leroux, L. Maino, G. Pope, D. Robert, B. Wesolowski (2024). *SQIsign2D-West: The Fast, the Small, and the Safer*. Cryptology ePrint Archive, Paper 2024/760, <https://eprint.iacr.org/2024/760>
- [2] L. De Feo, D. Kohel, A. Leroux, C. Petit, B. Wesolowski (2020). *SQISign: compact post-quantum signatures from quaternions and isogenies*. Cryptology ePrint Archive, Paper 2020/1240, <https://eprint.iacr.org/2020/1240>
- [3] L. De Feo, A. Leroux, P. Longa, B. Wesolowski (2022). *New algorithms for the Deuring correspondence: Towards practical and secure SQISign signatures*. Cryptology ePrint Archive, Paper 2022/234, <https://eprint.iacr.org/2022/234>
- [4] W. Castryck, T. Decru (2022). *An efficient key recovery attack on SIDH*. Cryptology ePrint Archive, Paper 2022/975, <https://eprint.iacr.org/2022/975>
- [5] P. Dartois, A. Leroux, D. Robert, B. Wesolowski (2023). *SQISignHD: New Dimensions in Cryptography*. Cryptology ePrint Archive, Paper 2023/436, <https://eprint.iacr.org/2023/436>
- [6] K. Nakagawa, H. Onuki (2024). *SQIsign2D-East: A New Signature Scheme Using 2-dimensional Isogenies*. Cryptology ePrint Archive, Paper 2024/771, <https://eprint.iacr.org/2024/771>
- [7] M. Duparc, T. Boris Fouotsa (2024). *SQIPrime: A dimension 2 variant of SQISignHD with non-smooth challenge isogenies*. Cryptology ePrint Archive, Paper 2024/773, <https://eprint.iacr.org/2024/773>
- [8] Intel Corporation. *Intel Intrinsics Guide*. Official Developer Documentation, <https://www.intel.com/content/www/us/en/docs/intrinsics-guide/index.html>, [Accessed 16th April, 2026]
- [9] M. A. Aardal, G. Adj, D. F. Aranha, A. Basso, I. Andrés Canales Martínez, J. Chávez-Saab, M. Corte-Real Santos, P. Dartois, L. De Feo, M. Duparc, J. Komada Eriksen, T. Boris Fouotsa, D. Luiz Gazzoni Filho, B. Hess, D. Kohel, A. Leroux, P. Longa, L. Maino, M. Meyer, K. Nakagawa, H. Onuki, L. Panny, S. Patranabis, C. Petit, G. Pope, K. Reijnders, D. Robert, F. Rodríguez-Henríquez, S. Schaeffler, B. Wesolowski (2025). *SQIsign*. National Institute of Standards and Technology, version 2.0.1, <https://sqisign.org/spec/sqisign-20250707.pdf>
- [10] J. Chávez-Saab, M. Corte-Real Santos, L. De Feo, J. Komada Eriksen, B. Hess, D. Kohel, A. Leroux, P. Longa, M. Meyer, L. Panny, S. Patranabis, C. Petit, F.

- Rodríguez-Henríquez, S. Schaeffler, B. Wesolowski (2023). *SQIsign*. National Institute of Standards and Technology, version 1.0, <https://sqisign.org/spec/sqisign-20230601.pdf>
- [11] National Institute of Standards and Technology (2024). *FIPS 204: Module-Lattice-Based Digital Signature Standard*. Federal Information Processing Standards (FIPS), <https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.204.pdf>
- [12] C. Turino, W. J. Buchanan, O. Lo, C. Thümmel (2025). *PQC-LEO: An Evaluation Framework for Post-Quantum Cryptographic Algorithms*. In 2025 IEEE 7th International Conference on Trust, Privacy and Security in Intelligent Systems, and Applications (TPS-ISA) 2025 Nov 12 (pp. 237-247). IEEE. <https://ieeexplore.ieee.org/abstract/document/11410195>
- [13] National Institute of Standards and Technology (2023). *Additional Post-Quantum Digital Signature Schemes*. Computer Security Resource Center (CSRC) Project Page, <https://csrc.nist.gov/projects/pqc-dig-sig>
- [14] R. L. Rivest, A. Shamir, L. Adleman (1978). *A Method for Obtaining Digital Signatures and Public-Key Cryptosystems*. Communications of the ACM, 21(2), 120–126, <https://doi.org/10.1145/359340.359342>
- [15] W. Diffie, M. Hellman (1976). *New Directions in Cryptography*. IEEE Transactions on Information Theory, 22(6), 644–654, <https://doi.org/10.1109/TIT.1976.1055638>
- [16] National Cyber Security Centre (2023). *Next steps in preparing for post-quantum cryptography*. NCSC White Paper, <https://www.ncsc.gov.uk/paper/next-steps-in-preparing-for-post-quantum-cryptography>
- [17] P. W. Shor (1994). *Algorithms for Quantum Computation: Discrete Logarithms and Factoring*. In Proceedings 35th Annual Symposium on Foundations of Computer Science, 124–134, <https://doi.org/10.1109/SFCS.1994.365700>
- [18] H. Adkins, S. Schmieg (2026). *Quantum frontiers may be closer than they appear*. Google Blog: Innovation & AI. March 25, 2026. <https://blog.google/innovation-and-ai/technology/safety-security/cryptography-migration-timeline/> [Accessed 20th April, 2026]
- [19] National Institute of Standards and Technology (2024). *What Is Post-Quantum Cryptography?*. August 13, 2024. <https://www.nist.gov/cybersecurity-and-privacy/what-post-quantum-cryptography> [Accessed 20th April, 2026]

- [20] National Institute of Standards and Technology (2016). *Post-Quantum Cryptography: Call for Proposals*. U.S. Department of Commerce, <https://csrc.nist.gov/CSRC/media/Projects/Post-Quantum-Cryptography/documents/call-for-proposals-final-dec-2016.pdf>
- [21] J. Komada Eriksen, L. Panny, J. Sotáková, M. Veroni (2023). *Deuring for the People: Supersingular Elliptic Curves with Prescribed Endomorphism Ring in General Characteristic Correspondence*. Norwegian University of Science and Technology. <https://jonathke.github.io/files/Deuring-for-the-people.pdf>
- [22] K. Eisenträger, S. Hallgren, K. Lauter, T. Morrison, C. Petit (2018). *Supersingular Isogeny Graphs and Endomorphism Rings: Reductions and Solutions*. Advances in Cryptology – EUROCRYPT 2018, 329–368, https://doi.org/10.1007/978-3-319-78372-7_11
- [23] D. Kohel, K. Lauter, C. Petit, J.P. Tignol (2014). *On the quaternion ℓ -isogeny path problem*. LMS Journal of Computation and Mathematics, 17(A), 418–432, <https://arxiv.org/pdf/1406.0981>
- [24] L. K. Grover (1996). *A fast quantum mechanical algorithm for database search*. In 28th ACM STOC. pp. 212–219. ACM Press. <https://doi.org/10.1145/237814.237866>
- [25] CRYSTALS Team. *CRYSTALS-Dilithium*. Official Project Website, <https://pq-crystals.org/dilithium/>
- [26] D. J. Bernstein, T. Lange. *SUPERCOP: System for Unified Performance Evaluation Related to Cryptographic Operations and Primitives*. <https://bench.cr.yp.to/supercop.html>
- [27] G. M. Amdahl (1967). *Validity of the single processor approach to achieving large scale computing capabilities*. Proceedings of the April 18-20, 1967, Spring Joint Computer Conference, 483–485, <https://doi.org/10.1145/1465482.1465560>
- [28] A. Singh, S. Kirillov (2021). *Intel Integrated Performance Primitives Cryptography Acceleration on 3rd Generation Intel Xeon Processor Scalable and 10th Gen Intel Core Processors*. Intel Developer Zone, Document ID: 659668. <https://www.intel.com/content/www/us/en/developer/articles/technical/intel-ipp-crypto-multi-buffer-acceleration.html>
- [29] T. Granlund and the GMP Development Team (2024). *GNU MP: The GNU Multiple Precision Arithmetic Library*. Version 6.3.0. <https://gmplib.org/gmp-man-6.3.0.pdf>

- [30] L. De Feo, S. D. Galbreith (2019). *SeaSign: Compact isogeny signatures from class group actions*. EUROCRYPT 2019 <https://eprint.iacr.org/2018/824.pdf>
- [31] National Institute of Standards and Technology (2026). *Post-Quantum Cryptography: Additional Digital Signature Schemes*. Computer Security Resource Center (CSRC) Project Page, <https://csrc.nist.gov/projects/pqc-dig-sig/round-3-additional-signatures>
- [32] J. H. Silverman (2009). *The Arithmetic of Elliptic Curves*. Graduate Texts in Mathematics, volume 106, 2nd edition, Springer.
- [33] L. De Feo (2017). *Mathematics of Isogeny Based Cryptography*. arXiv preprint arXiv:1711.04062, <https://arxiv.org/pdf/1711.04062>
- [34] A. Fiat, A. Shamir (1986). *How To Prove Yourself: Practical Solutions to Identification and Signature Problems*. In: Odlyzko, A.M. (eds) Advances in Cryptology — CRYPTO' 86. CRYPTO 1986. Lecture Notes in Computer Science, vol 263. Springer, Berlin, Heidelberg. https://doi.org/10.1007/3-540-47721-7_12
- [35] E. Kani (1997). *The number of curves of genus two with elliptic differentials*. Journal für die reine und angewandte Mathematik, Walter de Gruyter, pages 93-121 <https://mast.queensu.ca/~kani/papers/numgen1.pdf>
- [36] I. Damgård (2010). *On Σ -protocols*. Lecture Notes, University of Aarhus. <https://www.cs.au.dk/~ivan/Sigma.pdf>
- [37] J. Katz, Y. Lindell (2015). *Introduction to Modern Cryptography*. 2nd edition, CRC press. https://eclass.uniwa.gr/modules/document/file.php/CSCYB105/Reading%20Material/%5BJonathan_Katz%2C_Yehuda_Lindell%5D_Introduction_to_Mo%282nd%29.pdf
- [38] L. C. Washington (2008). *Elliptic Curves: Number Theory and Cryptography*. 2nd edition, Chapman and Hall/CRC
- [39] A. J. Menezes, P. Van Oorschot, S. A. Vanstone (1997). *Handbook of Applied Cryptography*. CRC Press
- [40] D. Mumford (1970). *Abelian Varieties*. Oxford University Press. https://wstein.org/edu/Fall2003/252/references/mumford-abvar/Mumford-Abelian_Varieties.pdf
- [41] SQISign Team, *the-sqisign (NIST Round 1)*, GitHub Repository. Available at: <https://github.com/SQISign/the-sqisign/tree/nist-v1>

- [42] SQISign Team, *the-sqisign (NIST Round 2)*, GitHub Repository. Available at: <https://github.com/SQISign/the-sqisign>
- [43] SQISign Team, *sqisign2d-west-ac24*, GitHub Repository. Available at: <https://github.com/SQISign/sqisign2d-west-ac24>
- [44] H. Onuki, *SQISign2D-East.jl*, GitHub Repository. Available at: <https://github.com/hiroshi-onuki/SQISign2D-East.jl>
- [45] P. Dartois, *SQISignHD-lib*, GitHub Repository. Available at: <https://github.com/Pierrick-Dartois/SQISignHD-lib>
- [46] The Sage Developers (2024). *SageMath, the Sage Mathematics Software System*. Available at: <https://www.sagemath.org>
- [47] J. Bezanson, A. Edelman, S. Karpinski, and V. B. Shah (2015), *Julia: A Fresh Approach to Numerical Computing*. <https://doi.org/10.48550/arXiv.1411.1607>
- [48] Anthropic, *Claude 5 Fable*, AI Large Language Model, 2026. Available at: <https://www.anthropic.com/claude/fable>.
- [49] Anthropic, *Claude 5 Opus*, AI Large Language Model, 2026. Available at: <https://www.anthropic.com/claude/opus>.
- [50] G. Goodwill, B. Jun, J. Jaffe, and P. Rohatgi (2011). *A testing methodology for side channel resistance validation*. In NIST non-invasive attack testing workshop. Available at: <https://www.rambus.com/wp-content/uploads/2015/08/a-testing-methodology-for-side-channel-resistance-validation.pdf>
- [51] D. Johnson (2021). *Apple M1 Microarchitecture Research*. Available: <https://dougallj.github.io/applecpu/firestorm.html>
- [52] A. Lubotzky (2011), *Expander graphs in pure and applied mathematics*, arXiv preprint arXiv:1105.2389, Available: <https://arxiv.org/abs/1105.2389>