computers

MDPI

Article

Benchmarking Post-Quantum Signatures and KEMs on General-Purpose CPUs Using a TCP Client–Server Testbed

Jesus Algar-Fernandez, Andrea Villacís-Vanegas, Ysabel Amaro-Aular and Maria-Dolores Cano *

Department of Information and Communication Technologies, Universidad Politécnica de Cartagena, 30202 Cartagena, Spain

* Correspondence: mdolores.cano@upct.es

Abstract

Quantum computing threatens widely deployed public-key cryptosystems, accelerating the adoption of Post-Quantum Cryptography (PQC) in practical systems. Beyond asymptotic security, the feasibility of PQC deployments depends on measured performance on real hardware and on implementation-level overheads. This paper presents an experimental evaluation of five post-quantum digital signature schemes (CRYSTALS-Dilithium, HAWK, SQISign, SNOVA, and SPHINCS+) and three key encapsulation mechanisms (Kyber, HQC, and BIKE) selected to cover multiple PQC design families and parameterizations used in practice. We implement a TCP client–server testbed in Python that invokes C implementations for each primitive—via standalone executables and, where provided, in-process dynamic libraries—and benchmarks key generation, encapsulation/decapsulation, and signature generation/verification on two Windows 11 commodity processors: an AMD Ryzen 7 4000 (8 cores, 16 threads, 1.8 GHz) and an Intel Core i5-1035G1 (4 cores, 8 threads, 1.0 GHz). Each operation is repeated ten times under a low-interference setup, and results are aggregated as mean (with 95% confidence intervals) timings over repeated runs. Across the evaluated configurations, lattice-based schemes (Kyber, Dilithium, HAWK) show the lowest computational cost, while code-based KEMs (HQC, BIKE), isogeny-based (SQISign), and multivariate (SNOVA) signatures incur higher overhead. Hash-based SPHINCS+ exhibits larger artifacts and higher signing latency depending on the parameterization. The AMD platform consistently outperforms the Intel platform, illustrating the impact of CPU characteristics on observed PQC overheads. These results provide comparative evidence to support primitive selection and capacity planning for quantum-resistant deployments, while motivating future end-to-end validation in protocol and web service settings.

Keywords: post-quantum cryptography (PQC); key encapsulation mechanisms (KEM); post-quantum digital signatures; ML-KEM (Kyber); ML-DSA (Dilithium); TLS 1.3/hybrid TLS; performance evaluation; cybersecurity; privacy; cryptography

Check for updates

Academic Editor: Paolo Bellavista

Received: 10 January 2026
Revised: 5 February 2026
Accepted: 6 February 2026
Published: 9 February 2026

Copyright: © 2026 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license.

1. Introduction

Large-scale quantum computing poses a direct threat to the public-key cryptosystems that currently secure internet communications, financial transactions, and critical infrastructures. Classical schemes such as RSA and Elliptic-Curve Cryptography (ECC) rely on the hardness of integer factorization and discrete logarithms, problems that can be solved in polynomial time on a quantum computer using Shor's algorithm [1]. Once cryptanalytically relevant quantum computers become available, today's public-key infrastructure

will no longer provide adequate security [2], motivating the design and deployment of Post-Quantum Cryptography (PQC).

Over the last decade, several mathematical families have emerged as leading candidates for PQC, including lattice-, code-, isogeny-, multivariate-, and hash-based constructions. A key milestone was NIST's report on Post-Quantum Cryptography (NISTIR 8105 [3]), which set out the initial threat assessment and transition considerations and announced the plan for an open standardization process. Comprehensive surveys such as [4] synthesize these algorithmic families and discuss directions for the transition process. After a multi-year public evaluation process, the U.S. National Institute of Standards and Technology (NIST) selected CRYSTALS-Kyber for key establishment and CRYSTALS-Dilithium, Falcon, and SPHINCS+ for digital signatures. NIST has since published the first Federal Information Processing Standards (FIPS) for these schemes: ML-KEM (FIPS 203) [5], corresponding with CRYSTALS-Kyber; ML-DSA (FIPS 204) [6], corresponding with CRYSTALS-Dilithium; and SLH-DSA (FIPS 205) [7], corresponding with SPHINCS+. Falcon is being standardized separately as FN-DSA in FIPS 206 (currently in development). In parallel, additional proposals such as HQC [8] and BIKE [9] have been kept under consideration to provide algorithmic diversity in case new attacks appear on structured lattices.

While the asymptotic security of these schemes has been analyzed extensively, their practical performance on real platforms remains a critical factor for adoption. Implementations must cope with larger keys and signatures, different memory access patterns, and more complex arithmetic than traditional RSA/ECC, which can stress constrained devices and high-throughput servers alike. Recent surveys and measurement studies emphasize that deployment decisions hinge not only on security guarantees but also on execution time, memory footprint, throughput, and communication overhead in specific environments. At the same time, a broader literature is emerging on how quantum computing interacts with cybersecurity and sustainability in industrial systems, analyzing its role in next-generation quantum security architectures [2], in circular economy and Industry 4.0 settings [10], and in energy efficiency and environmental performance [11]. In parallel, recent work has begun to measure the impact of PQC on constrained devices and Industrial Internet of Things (IIoT) scenarios [12], where the computational and communication overhead of new primitives is particularly critical. These perspectives reinforce the importance of understanding the concrete resource costs associated with PQC adoption.

A growing body of work has begun to quantify these costs. In [13], Abbasi et al. present a cross-platform benchmark of NIST-selected algorithms such as Kyber, Dilithium, and BIKE across heterogeneous computing environments, measuring latency, memory usage, and protocol overhead at multiple security levels. Other studies evaluate PQC in specific protocol contexts: Paquin et al. [14] and Sikeridis et al. [15] analyze the impact of hybrid and post-quantum primitives on TLS 1.3 handshakes, Montenegro et al. [16] propose a performance evaluation framework for post-quantum TLS configurations combining classical and PQC ciphersuites [16], and Juaristi et al. [17] benchmark PQC in Ethereum-based blockchains, focusing on transaction-level costs. Complementary work targets resource-constrained platforms, assessing the feasibility of PQC on microcontrollers and IoT devices and measuring the impact of PQC in IIoT scenarios where computational and communication overheads may be particularly restrictive. These contributions collectively show that performance varies substantially across algorithm families, parameter sets, and deployment scenarios and that no single primitive dominates along all dimensions. Recent experimental work also emphasizes that primitive-level costs and deployment-level costs can diverge substantially once post-quantum schemes are integrated into real protocols and services. In particular, hybrid TLS 1.3 deployments that combine classical and post-

quantum mechanisms (and, in some proposals, additional key material sources) show that operational overhead depends not only on the primitive's standalone cost but also on how it is integrated into the handshake and the surrounding protocol logic [18]. Complementary TLS-oriented analyses [19] further illustrate how updated post-quantum standardization outcomes and candidate diversity (e.g., the inclusion of HQC as a key establishment standard alongside ongoing evaluation of code-based alternatives) motivate continuous performance validation in deployment-relevant settings.

Despite this progress, there is still value in controlled, platform-level benchmarks that (i) compare multiple families of key encapsulation mechanisms (KEM) and signature schemes under a unified methodology, (ii) use commodity general-purpose processors representative of client and server machines, and (iii) include both standardized algorithms and research-stage proposals that may influence future designs. Particularly, lattice-based schemes, such as Kyber, Dilithium, and HAWK; code-based KEMs, such as HQC and BIKE; isogeny-based signatures, such as SQISign; multivariate-based signatures, such as SNOVA; and hash-based SPHINCS+ offer very different trade-offs between key and signature size, computational cost, and implementation complexity. A comparative evaluation of these schemes on typical desktop-class CPUs, in a networked setting, can help bridge the gap between algorithm design and system deployment.

This paper contributes to that effort with an experimental evaluation of eight post-quantum primitives: three key encapsulation mechanisms (Kyber-FIPS 203, HQC, and BIKE) and five digital signature schemes (CRYSTALS-Dilithium-FIPS 204, HAWK, SQISign, SNOVA, and SPHINCS+-FIPS 205). We implement a TCP client–server testbed in Python that invokes C executables for each primitive and measure key generation, encapsulation/decapsulation, and signature generation/verification times on two commodity processors: an AMD Ryzen 7 4000 (8 cores, 16 threads, 1.8 GHz) and an Intel Core i5-1035G1 (4 cores, 8 threads, 1.0 GHz), both running Windows 11. Each operation is executed ten times under controlled load, and the results are aggregated to obtain stable performance indicators.

The main contributions of this work are as follows:

1. A unified benchmarking framework for post-quantum key encapsulation mechanisms (KEM) and digital signatures in a client–server setting, suitable for evaluating networked cryptographic operations on general-purpose processors;
2. A comparative performance study of five signature schemes and three KEMs spanning lattice-based, code-based, isogeny-based, multivariate-based, and hash-based families, using consistent metrics (key generation, encapsulation/decapsulation, signing, verification, key sizes) across two hardware platforms;
3. An analysis of algorithm–platform interactions, highlighting how architectural differences between AMD Ryzen 7 4000 and Intel Core i5-1035G1 influence the relative cost of PQC operations and the practicality of different schemes for deployment in networked information systems.

Prior benchmarking studies often focus on only NIST-selected schemes, a single protocol embedding (e.g., TLS), or constrained platforms (IoT/microcontrollers). In contrast, the goal of this study is a like-for-like comparison of both standardized baselines (e.g., Kyber/ML-KEM, Dilithium/ML-DSA, SPHINCS+/SLH-DSA) and research-stage designs (e.g., BIKE, HQC, HAWK, SQISign, SNOVA) executed under the same orchestration pipeline, measurement boundary, and reporting format. This consolidated view is intended to help practitioners understand how non-standardized alternatives compare to today's standardization baseline on mainstream Windows client/server hardware, including both timing and artefact-size trade-offs. The insights derived from these measurements complement existing PQC benchmarking and protocol integration studies and provide empirical

guidance for practitioners selecting and implementing quantum-resistant primitives on mainstream hardware.

From a practical standpoint, the resulting benchmark data support two immediate needs in quantum-resistant planning: first, evidence-based primitive selection under a consistent measurement boundary across multiple PQC families and parameter sets, and second, capacity planning on mainstream client/server hardware by showing how observed overheads vary across operations and across representative CPU platforms. These outcomes are intended to help practitioners anticipate computational and artefact-size trade-offs when integrating post-quantum primitives into networked services.

The rest of this article is organized as follows. Section 2 reviews the relevant background and related work on post-quantum key encapsulation mechanisms and digital signature schemes. Section 3 describes the experimental methodology, including the client–server testbed, hardware platforms, and performance metrics. Section 4 presents and analyzes the benchmarking results for the evaluated KEMs and signature schemes on both processors. Section 5 discusses the main findings, practical implications, and limitations of this study. Section 6 concludes this paper and outlines directions for future work.

2. Background and Related Work

2.1. Quantum Computing and Its Impact on Cryptography

Quantum computing has introduced a paradigm shift in computational capabilities. By exploiting quantum superposition and entanglement, quantum devices can, in principle, solve specific problems much more efficiently than classical computers. One of the most critical implications for information security is Shor's algorithm, which can solve integer factorization and discrete logarithm problems in polynomial time on a sufficiently powerful quantum computer. As a result, widely deployed public-key schemes such as RSA and ECC would become vulnerable once cryptanalytically relevant quantum computers are available.

This threat motivates a transition towards cryptographic schemes that are secure against both classical and quantum adversaries. The challenge is not only mathematical but also practical: new schemes must be integrated into existing protocols and infrastructures without unacceptable performance or interoperability penalties.

2.2. Post-Quantum Cryptography and the NIST Standardization Process

Recognizing the vulnerabilities posed by quantum computing, NIST launched a public process to standardize PQC. NISTIR 8105 outlined the initial threat assessment and transition considerations and announced the plan for an open evaluation of candidate schemes. Over several rounds, NIST assessed submissions based on security, performance, implementation characteristics, and suitability for different applications.

As summarized in recent surveys such as [4], five main mathematical families have emerged as leading candidates: lattice-based cryptography (e.g., Kyber, CRYSTALS-Dilithium, Falcon), code-based cryptography (e.g., HQC, BIKE), isogeny-based cryptography (e.g., SQISign), multivariate-based cryptography (e.g., SNOVA), and hash-based cryptography (e.g., SPHINCS+).

After a multi-year public evaluation process, NIST selected CRYSTALS-Kyber as the primary key establishment mechanism and CRYSTALS-Dilithium, Falcon, and SPHINCS+ as digital signature schemes and has published the first Federal Information Processing Standards (FIPS) for these algorithms (ML-KEM, ML-DSA, and SLH-DSA, respectively). Falcon has also been selected and is being standardized separately as FN-DSA in FIPS 206 (currently in development). Additional proposals, such as HQC and BIKE, remain under consideration to provide algorithmic diversity in case new structural weaknesses are found in lattice-based designs. Each family presents distinct trade-offs in terms of key and

ciphertext size, signature size, computational complexity, and implementation constraints, which makes empirical performance evaluation essential for deployment decisions.

2.3. Post-Quantum Key Encapsulation Mechanisms

KEMs are crucial for establishing shared secrets over untrusted networks. In the PQC context, lattice-based and code-based constructions are among the most prominent KEM families.

Kyber is a lattice-based KEM built on the Module-LWE problem. It was selected by NIST as the primary standard for key establishment due to its strong security arguments, relatively compact keys and ciphertexts, and efficient implementations on a wide range of platforms. Its structured lattice design allows for vectorized and constant-time implementations, making it attractive for high-performance servers and constrained devices alike.

HQC and BIKE are code-based KEMs derived from the McEliece tradition. They use quasi-cyclic codes to reduce key sizes compared with classical code-based proposals while retaining conservative security assumptions. HQC relies on hard decoding problems for quasi-cyclic codes in the Hamming metric (syndrome decoding), whereas BIKE is based on the hardness of decoding in moderate-density parity-check codes. Both schemes provide valuable algorithmic diversity but typically exhibit larger public keys or higher computational cost than lattice-based KEMs, which motivates careful performance benchmarking on different hardware platforms.

2.4. Post-Quantum Digital Signatures

Digital signatures provide authenticity, integrity, and non-repudiation and are a critical component of public-key infrastructures, software update mechanisms, and many authentication protocols. In the post-quantum setting, several families of signature schemes have been proposed and analyzed.

First, CRYSTALS-Dilithium is a lattice-based signature scheme built on Module-LWE and Module-SIS problems. It was standardized by NIST as ML-DSA and is widely regarded as a strong default choice due to its favorable balance between security, signature size, and computational efficiency. Second, SPHINCS+ is a stateless hash-based signature scheme that relies only on the security of underlying hash functions. It offers strong long-term security and robustness against advances in number theoretic cryptanalysis at the cost of relatively large signatures and higher signing time, especially at higher security levels. Third, SQISign is an isogeny-based signature scheme that achieves very compact signatures and public keys by leveraging hard problems in the isogeny graphs of supersingular elliptic curves or abelian varieties. However, they typically entail higher computational cost and more complex implementations than lattice-based designs. Fourth, SNOVA is a multivariate signature scheme. Fifth and last, HAWK is a lattice-based deterministic signature scheme designed to produce compact signatures while maintaining strong security guarantees. Its structure and parameter choices differ from Dilithium, providing an additional design point within the lattice-based family.

These schemes illustrate the diversity of design approaches and efficiency profiles in post-quantum digital signatures, underscoring the need for comparative performance studies under realistic conditions.

2.5. Performance Evaluation of PQC Implementations

Beyond asymptotic complexity, the adoption of PQC depends critically on measured performance in specific environments. Recent work has begun to quantify the cost of PQC primitives in various deployment scenarios.

Abbasi et al. [13] present a cross-platform benchmark of NIST-selected algorithms such as Kyber, Dilithium, and BIKE across heterogeneous computing environments, mea-

suring latency, memory usage, and protocol overhead at multiple security levels. Paquin et al. [14] studied the impact of post-quantum and hybrid key exchange and authentication mechanisms on TLS 1.3 handshakes, providing empirical data for protocol-level design choices. In [16], Montenegro et al. propose a performance evaluation framework for post-quantum TLS configurations combining classical and PQC cipher suites, and Juaristi et al. [17] benchmark PQC in Ethereum-based blockchains, focusing on transaction-level costs and throughput.

Complementary studies analyze PQC on microcontrollers and IoT devices, as well as in IIoT contexts, e.g., [12], where CPU cycles, memory, and bandwidth are limited, and even moderate overheads may be problematic. In parallel, broader work on quantum computing and cybersecurity examines how quantum-safe primitives interact with sustainability, energy efficiency, and Industry 4.0 architectures (e.g., [2,10,11]).

These studies collectively show that performance varies substantially across algorithm families, parameter sets, platforms, and protocol embeddings and that no single primitive is optimal along all dimensions. However, there remains a need for controlled, platform-level benchmarks that directly compare multiple KEMs and signature schemes on commodity general-purpose processors under a unified methodology, which is precisely the focus of the experimental study presented in this paper.

In this sense, our measurements complement (rather than replace) both microbenchmarks that isolate kernel-level costs and protocol-level studies (e.g., TLS) that include full handshake and network effects. Specifically, our results quantify a deployment-proximate invocation boundary for primitive operations under a unified orchestration workflow on commodity CPUs while explicitly excluding TCP transmission and request parsing/serialization, as defined next in Section 3.2.

3. Methodology

This section describes the experimental design used to evaluate the performance of selected post-quantum KEMs and digital signature schemes on commodity general-purpose processors, using a unified measurement procedure and consistent metrics.

3.1. Evaluated Primitives and Measured Operations

We evaluate three KEMs, namely, Kyber, HQC, and BIKE, and five digital signature schemes, namely, CRYSTALS-Dilithium, HAWK, SQISign, SNOVA, and SPHINCS+ (see Table 1). For KEMs, we measure the execution time of the following operations:

- Key generation: generation of a public/secret key pair;
- Encapsulation (encryption): generation of a ciphertext and shared secret using the public key;
- Decapsulation (decryption): derivation of the shared secret from the ciphertext using the secret key.

For digital signatures, we measure the following:

- Key generation: generation of a public/secret signing key pair;
- Signature generation (signing): creation of a signature over an input message;
- Signature verification: verification of a signature using the corresponding public key.

In addition to execution time, we include size-related indicators (where available through the invoked implementations) for the main artefacts exchanged or stored by each primitive (e.g., public keys, secret keys, ciphertexts, and signatures). These size indicators are used to support the discussion of practical deployment trade-offs together with timing results.

Table 1. Evaluated configurations.

Primitive	Type	Family	Configurations Evaluated (Labeled in Plots)	Notes
Kyber	KEM	Lattice-based	Kyber-512, Kyber-768, Kyber-1024 Kyber-512-90s, Kyber-768-90s, Kyber-1024-90s	Kyber-512/768/1024 are the standard Kyber parameter sets, and "-90s" denotes the AES- and SHA2-based variants defined in the Kyber specification (as opposed to SHAKE-based hashing)
BIKE	KEM	Code-based	LEVEL1, LEVEL2, LEVEL3	LEVEL1–LEVEL3 denote the BIKE parameter sets used in this study, representing increasing security/performance trade-offs as defined by the BIKE implementation
HQC	KEM	Code-based	128, 192, 256	128/192/256 denote the HQC parameter sets used in this study, representing increasing security/performance trade-offs as defined by the HQC implementation
CRYSTALS-Dilithium	Signature	Lattice-based	2, 3, 5; 2aes, 3aes, 5aes	Dilithium2/3/5 are the standard parameter sets, and "2aes/3aes/5aes" denote variants that use AES-256 in counter mode for expansion instead of SHAKE-based hashing, as specified in the Dilithium design
HAWK	Signature	Lattice-based	256, 512, 1024	256/512/1024 denote the HAWK parameter sets used in this study, representing increasing security/performance trade-offs as defined by the HAWK implementation
SQISign	Signature	Isogeny-based	LOW2, MEDIUM2, HIGH2	LOW2/MEDIUM2/HIGH2 denote the SQISign parameter sets used in this study, representing increasing security levels as defined by the SQISign implementation
SNOVA	Signature	Multivariate	LOW1, MEDIUM1, HIGH1	LOW1/MEDIUM1/HIGH1 denote the SNOVA parameter sets used in this study, representing increasing security levels as defined by the SNOVA implementation
SPHINCS+	Signature	Hash-based	128s, 128f, 192s, 192f, 256s, 256f	128/192/256 denote the SPHINCS+ security levels, and "s" and "f" denote the 'small' and 'fast' variants, respectively, which trade signature size against computation

3.2. Testbed Architecture and Implementation Approach

To orchestrate experiments in a way that resembles typical deployment patterns (client requesting cryptographic services from a server-side component), we implement a lightweight TCP client–server testbed.

A client component initiates requests specifying the primitive (e.g., Kyber, Dilithium) and the operation type (e.g., key generation, encapsulation, signing). Then, a server component performs the requested operation and returns the results needed to complete the

measurement workflow (e.g., produced keys, ciphertexts, signatures, and/or verification outcomes depending on the operation).

The testbed is implemented in Python as the orchestration layer. The actual cryptographic computations are performed by invoking compiled implementations corresponding to each evaluated primitive. Depending on the available implementation, primitives are executed either as standalone binaries (invoked via *subprocess.run(...)*) or as in-process dynamic libraries (DLLs) accessed via *ctypes*. This design keeps the orchestration code consistent across schemes while allowing each primitive to be executed using its own implementation. The testbed code is available at [20].

For reproducibility, in addition to providing the source code, we define the measurement boundary as follows. For each request, the client submits a primitive identifier, parameterization identifier, and operation type to the server. The server triggers the corresponding compiled executable and records the wall-clock duration from immediately before invoking the operation (i.e., right before calling the executable through *subprocess.run(...)* or the corresponding DLL routine) to immediately after it returns (i.e., when the subprocess exits/the DLL call returns). This timing excludes TCP send/receive overhead and request parsing/serialization; it is taken after the request has been received and inputs are prepared. For executable-based schemes, it includes process start-up and the complete execution of the invoked binary; for DLL-based schemes, it measures only the duration of the library routine call. For signature generation/verification, the workload consists of signing/verifying a fixed-length message of 4 bytes generated as the UTF-8 encoding of the ASCII string "hola" (written to *message.txt*). For KEM operations, the workflow follows the standard sequence key generation–encapsulation–decapsulation, and correctness is verified by comparing derived shared secrets. All reported results correspond to the parameterizations configured in the evaluated executables summarized in Table 1.

3.3. Hardware and Operating System Environment

All experiments are conducted on two commodity platforms representative of widely deployed client/server machines: (1) AMD Ryzen 7 4000 series (8 cores/16 threads, base frequency of 1.8 GHz), Windows 11, and (2) Intel Core i5-1035G1 (4 cores/8 threads, base frequency of 1.0 GHz), Windows 11. Details are included in Table 2.

Table 2. Experimental platform and software configuration.

Feature	**Values**	
Platform	**AMD**	**Intel**
CPU model	AMD Ryzen 7 4000 series	Intel Core i5-1035G1
Cores/threads	8 cores/16 threads	4 cores/8 threads
Base/boost frequency	1.8 GHz	1.0 GHz
RAM (GB)	16 GB	8 GB
OS edition—version/build	Windows 11 Home (kernel 10.0.26100)	Windows 11 Home (kernel 10.0.26100)
Python version	CPython 3.12	CPython 3.12
Compiler/toolchain used for executables	GNU toolchain via MSYS2 MinGW-w64 (compiler version 14.2.0)	GNU toolchain via MSYS2 MinGW-w64 (compiler version 14.2.0)
Compilation flags	-O3	-O3
Power plan/CPU frequency scaling configuration	Windows default	Windows default

Using two distinct CPUs allows us to observe how the relative cost of PQC operations varies with architectural characteristics (e.g., core count and base frequency) under the same operating system family and measurement tooling.

3.4. Timing Methodology and Execution Procedure

We report wall-clock execution time (seconds) for each primitive, parameterization, and operation type (key generation, encapsulation/decapsulation for KEMs, and signature generation/verification for signature schemes). Timing is recorded locally at the endpoint that executes the cryptographic operation inside our TCP client–server testbed (default loopback configuration 127.0.0.1 in the scripts), so network transmission time is not part of the reported cryptographic timings.

The timing boundary depends on how the primitive is integrated in the testbed. For schemes executed via standalone binaries, the orchestration layer measures elapsed time as the difference between timestamps taken immediately before and after the corresponding *subprocess.run(...)* call (implemented using *time.time()* or *time.time()**1000 in the scripts). This boundary includes process start-up and the work performed inside the executable but excludes any preprocessing performed outside the timed region (e.g., preparing input files before launching the process). For schemes accessed as in-process dynamic libraries (e.g., via *ctypes.CDLL*/*ctypes.WinDLL*), elapsed time is measured around the specific library function call using *time.perf_counter()*, capturing the cost of the cryptographic routine itself. In the SNOVA scripts specifically, the message hashing step (SHAKE256) is computed outside the timed interval, and the recorded signing/verification times correspond to the core signing/verification routines over the derived digest. Because some primitives are invoked via standalone executables and others via in-process DLL calls, absolute timings should be interpreted within this measurement boundary; relative comparisons are primarily meaningful within each integration class.

Each experiment is repeated ten times on each platform. We report the mean, standard deviation, and 95% confidence intervals in Figures 1–12 and Tables 3 and 4, respectively, and visualize per-operation comparisons across schemes.

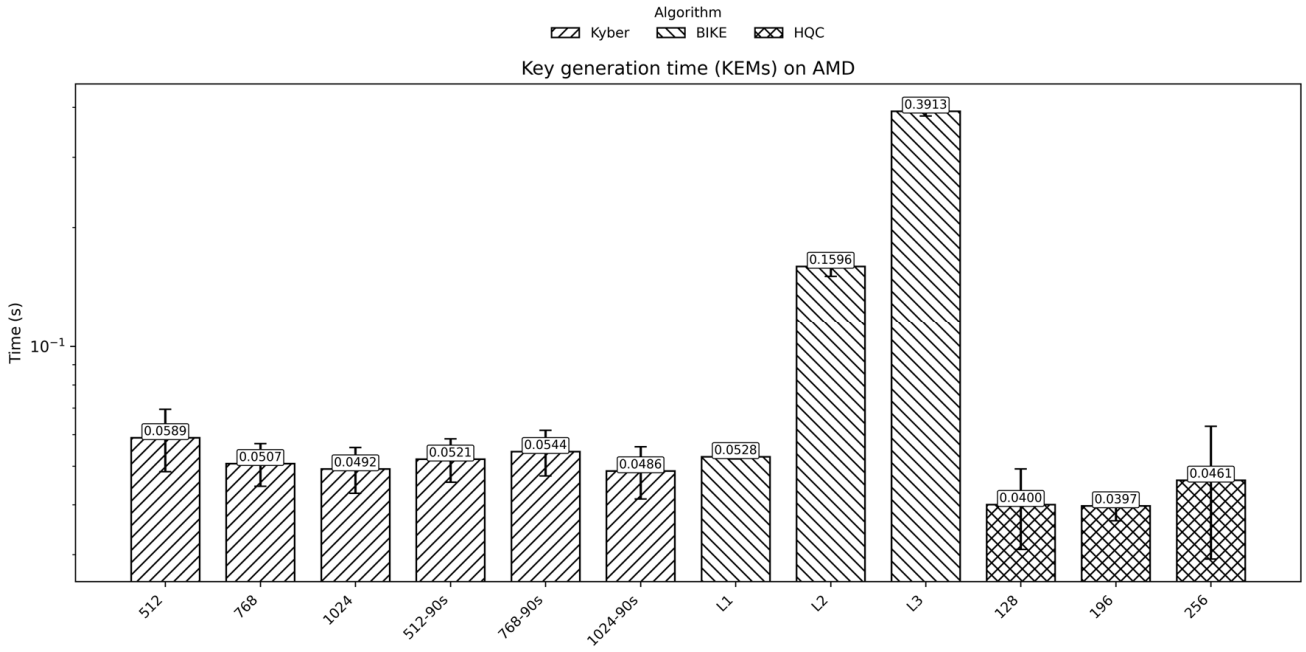

Figure 1. Key generation time (mean over 10 repetitions ± standard deviation) for Kyber, BIKE, and HQC on AMD.

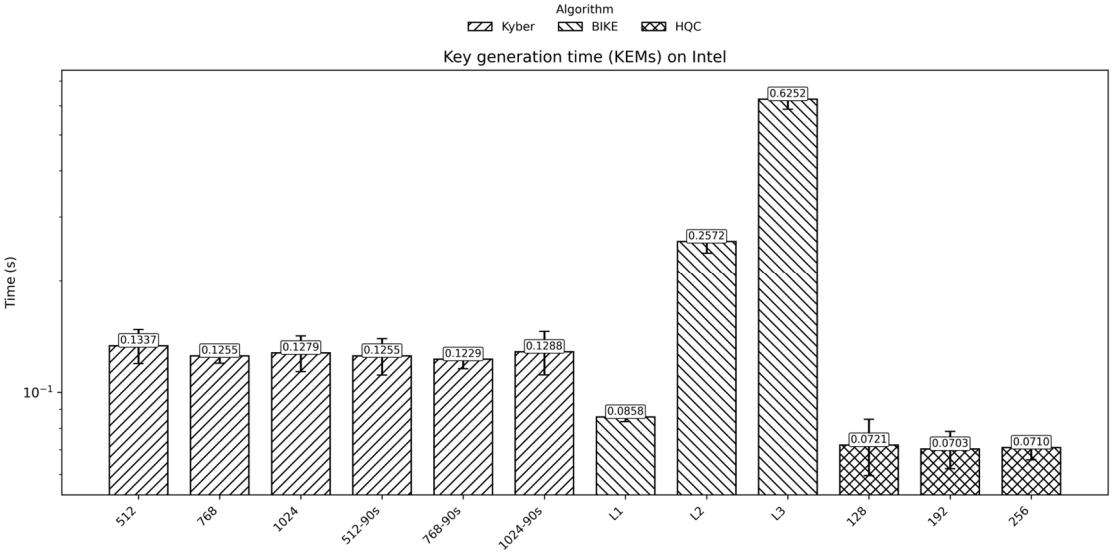

Figure 2. Key generation time (mean over 10 repetitions ± standard deviation) for Kyber, BIKE, and HQC on Intel.

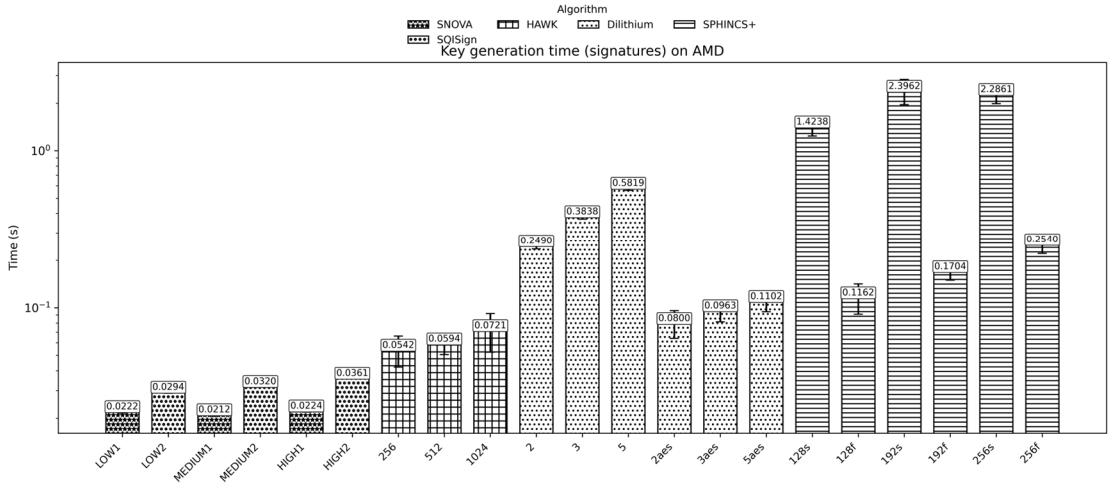

Figure 3. Key generation time (mean over 10 repetitions ± standard deviation) for SPHINCS+, SNOVA, SQISign, Hawk, and Dilithium on AMD.

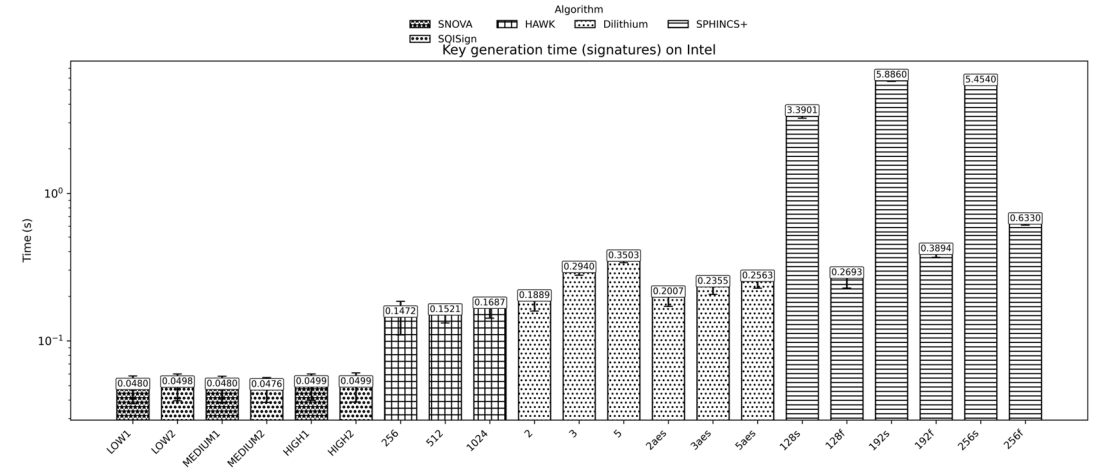

Figure 4. Key generation time (mean over 10 repetitions ± standard deviation) for SPHINCS+, SNOVA, SQISign, Hawk, and Dilithium on Intel.

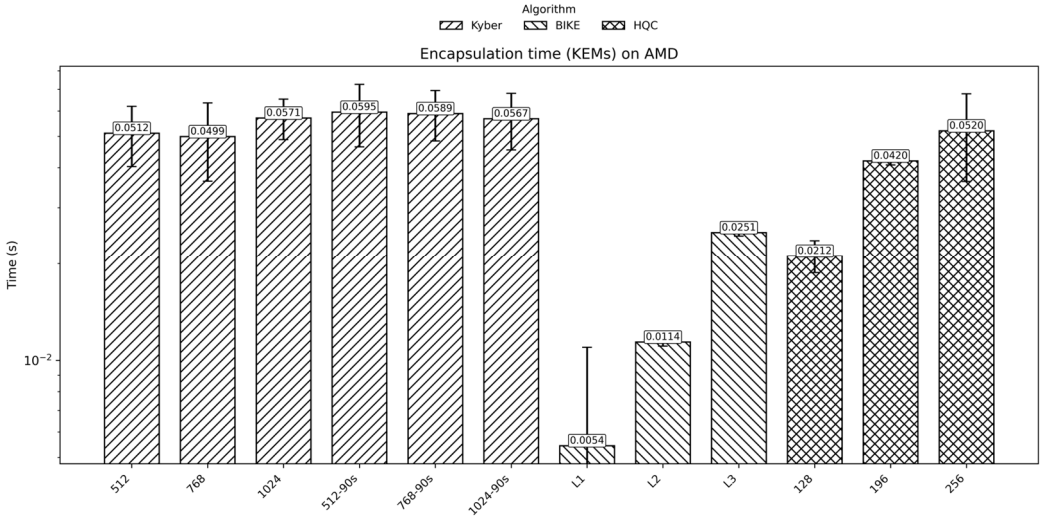

Figure 5. Encryption time (mean over 10 repetitions ± standard deviation) for Kyber, BIKE, and HQC on AMD.

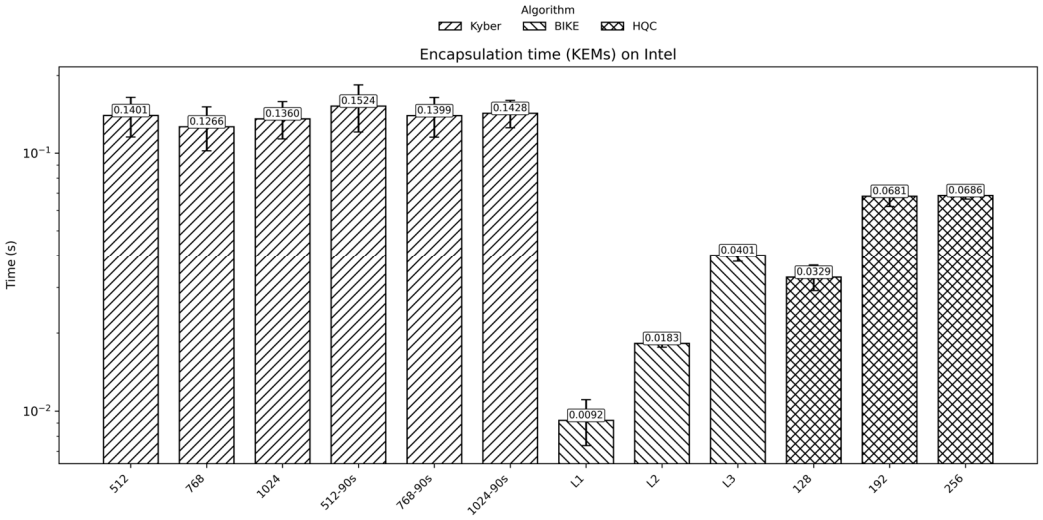

Figure 6. Encryption time (mean over 10 repetitions ± standard deviation) for Kyber, BIKE, and HQC on Intel.

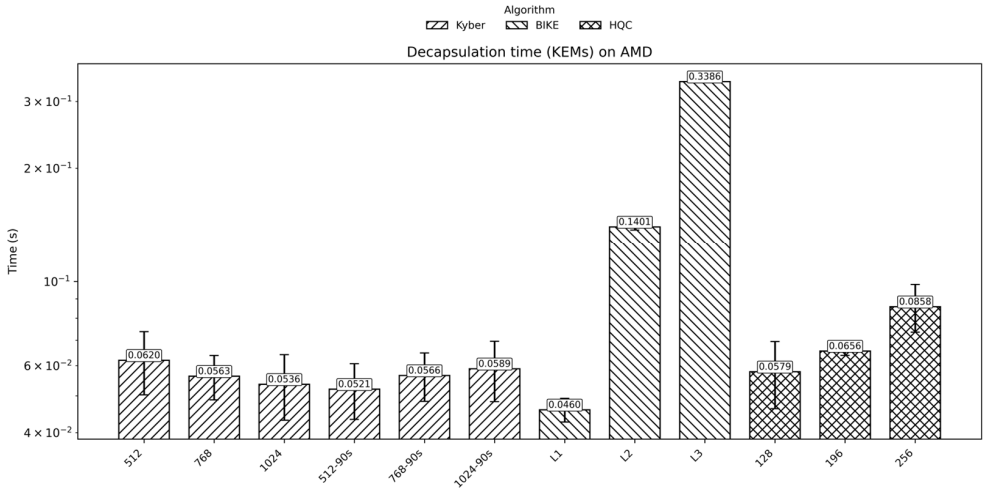

Figure 7. Decryption time (mean over 10 repetitions ± standard deviation) for Kyber, BIKE, and HQC on AMD.

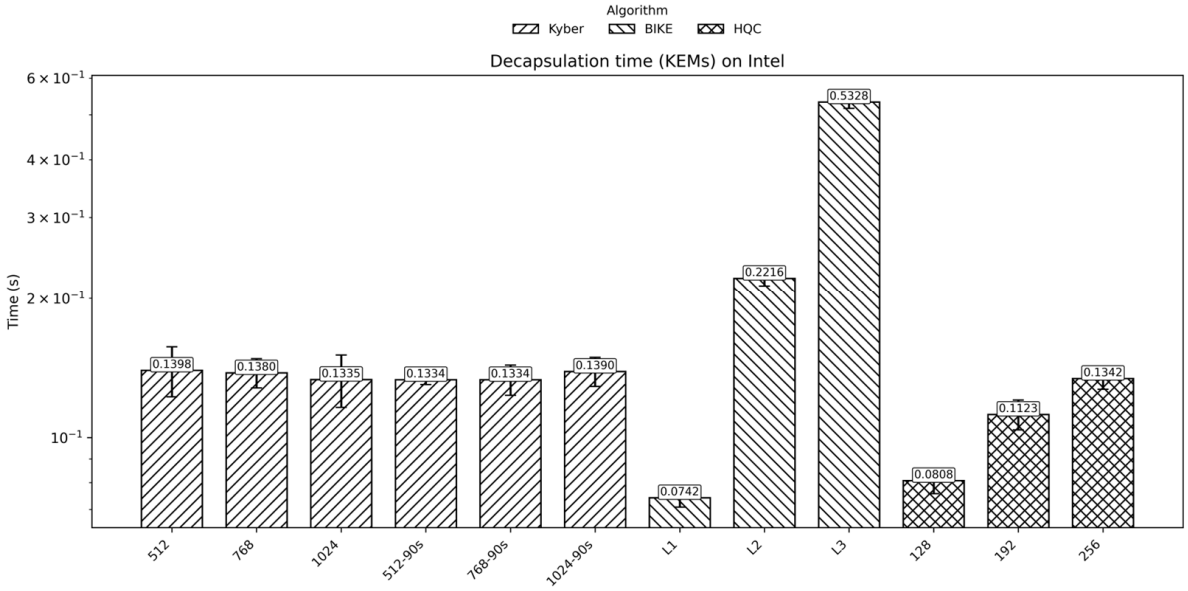

Figure 8. Decryption time (mean over 10 repetitions ± standard deviation) for Kyber, BIKE, and HQC on Intel.

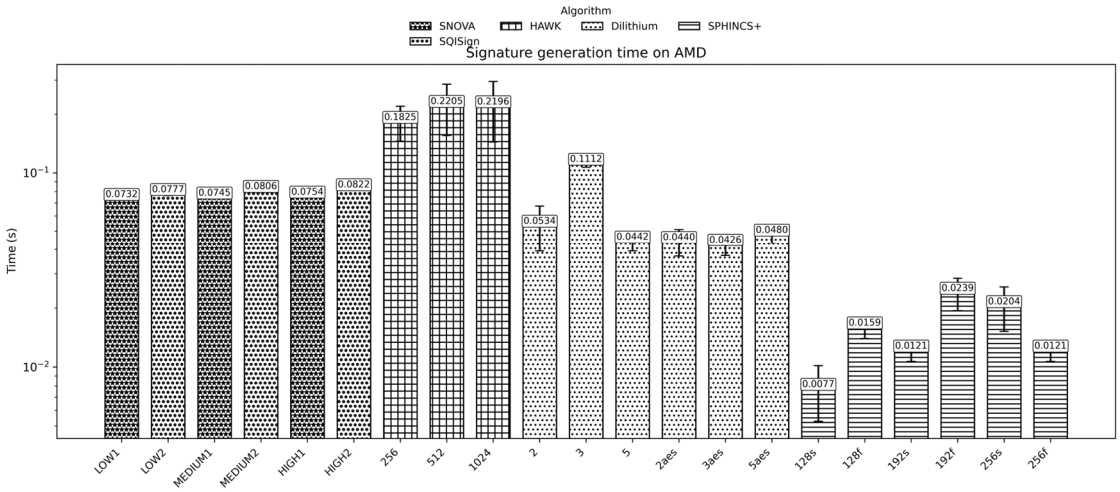

Figure 9. Signature generation time (mean over 10 repetitions ± standard deviation) for SPHINCS+, SNOVA, SQISign, Hawk, and Dilithium on AMD.

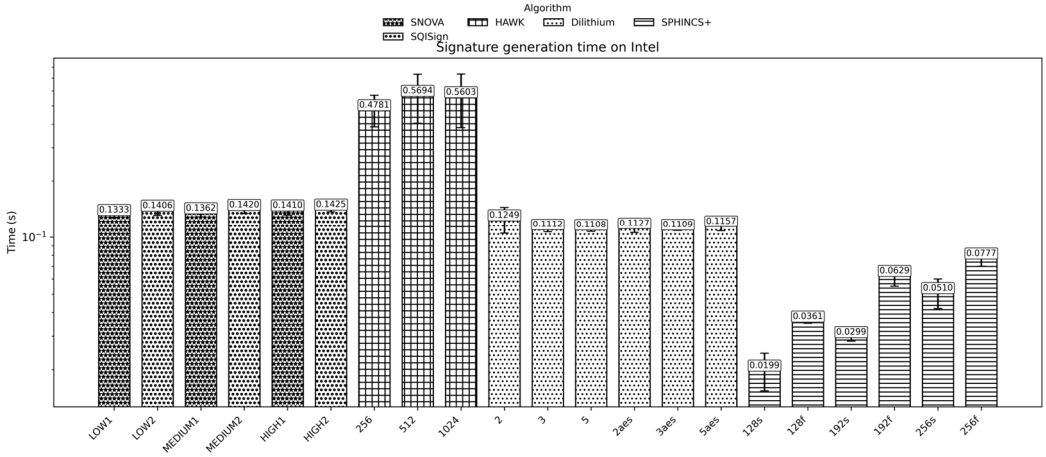

Figure 10. Signature generation time (mean over 10 repetitions ± standard deviation) for SPHINCS+, SNOVA, SQISign, Hawk, and Dilithium on Intel.

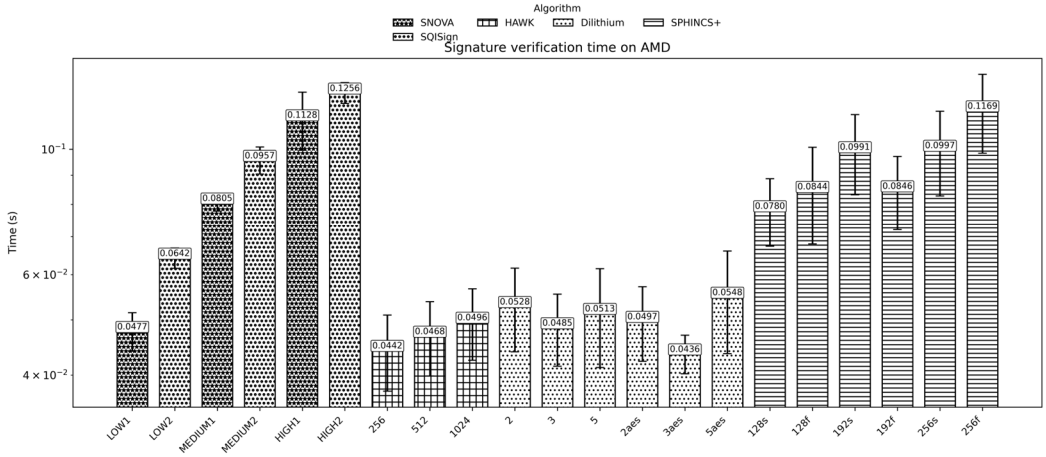

Figure 11. Signature verification time (mean over 10 repetitions ± standard deviation) for SPHINCS+, SNOVA, SQISign, Hawk, and Dilithium on AMD.

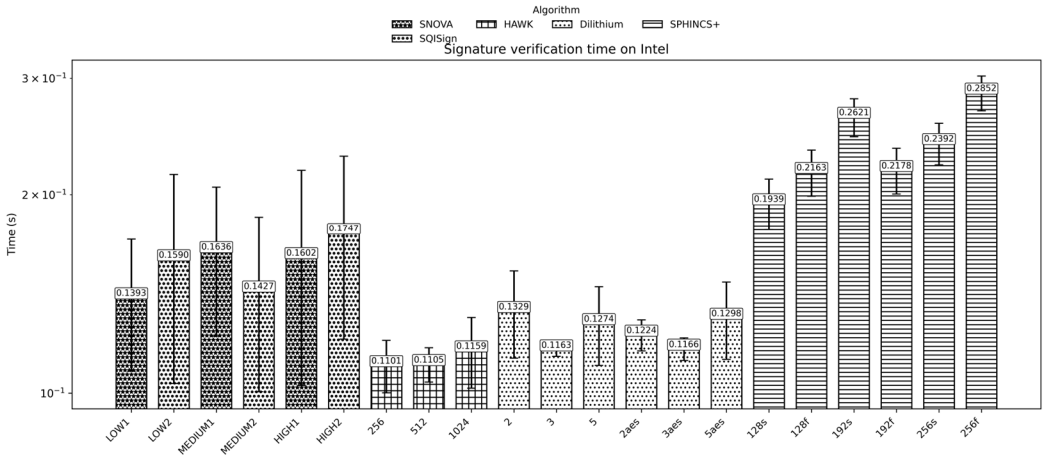

Figure 12. Signature verification time (mean over 10 repetitions ± standard deviation) for SPHINCS+, SNOVA, SQISign, Hawk, and Dilithium on Intel.

Table 3. Measurements (mean over 10 repetitions and 95% confidence intervals) for key generation, KEM encapsulation, and KEM decapsulation.

Algorithm	AMD						Intel					
	Key Generation (s)		KEM Encapsulation (s)		KEM Decapsulation(s)		Key Generation (s)		KEM Encapsulation (s)		KEM Decapsulation (s)	
	Mean	±CI	Mean	±CI	Mean	±CI	Mean	±CI	Mean	±CI	Mean	±CI
Kyber 512	0.0589	0.0079	0.0512	0.0082	0.062	0.0089	0.1337	0.0116	0.1402	0.0185	0.1398	0.0139
Kyber 768	0.0507	0.0047	0.0499	0.0103	0.0563	0.0057	0.1255	0.0043	0.1266	0.0181	0.138	0.0076
Kyber 1024	0.0492	0.0049	0.0571	0.0062	0.0536	0.0079	0.1276	0.0103	0.136	0.0164	0.1335	0.0136
Kyber 512-90s	0.0521	0.0049	0.0595	0.0099	0.0521	0.0066	0.1255	0.0103	0.1524	0.0235	0.1334	0.0028
Kyber 768-90s	0.0544	0.0054	0.0589	0.0079	0.0566	0.0062	0.1229	0.0053	0.1399	0.0178	0.1334	0.009
Kyber 1024-90s	0.0486	0.0055	0.0567	0.0086	0.0589	0.0081	0.1288	0.0126	0.1428	0.0136	0.139	0.0074
BIKE Level1	0.0528	0.0005	0.0058	0.0009	0.0451	0.0008	0.0858	0.0019	0.0092	0.0014	0.0742	0.0025
BIKE Level2	0.1596	0.0068	0.0114	0.0003	0.1401	0.0021	0.2572	0.0138	0.0183	0.0005	0.2216	0.0063
BIKE Level3	0.3913	0.008	0.0251	0.0005	0.3386	0.0012	0.6252	0.0291	0.0401	0.0017	0.5328	0.0124
HQC 128	0.04	0.0069	0.0212	0.0019	0.0579	0.0088	0.0721	0.0094	0.033	0.0028	0.0808	0.0038
HQC 192	0.0397	0.0028	0.042	0.0009	0.0656	0.0013	0.0703	0.0061	0.0682	0.0045	0.1123	0.0063
HQC 256	0.0461	0.0127	0.052	0.0119	0.0858	0.0093	0.071	0.004	0.0686	0.0017	0.1342	0.0053

Table 4. Measurements (mean over 10 repetitions and 95% confidence intervals) signature generation and verification.

Algorithm	AMD				Intel			
	Signature Generation (s)		Signature Verification (s)		Signature Generation (s)		Signature Verification (s)	
	Mean	±CI	Mean	±CI	Mean	±CI	Mean	±CI
SPHINCS+ 128s	0.0077	0.0018	0.078	0.0081	0.0199	0.0031	0.1939	0.0131
SPHINCS+ 128f	0.0159	0.0014	0.0844	0.0124	0.0361	0.0008	0.2163	0.013
SPHINCS+ 192s	0.0121	0.0011	0.0991	0.012	0.0299	0.0012	0.2621	0.0123
SPHINCS+ 192f	0.0239	0.0034	0.0846	0.0095	0.0629	0.0059	0.2178	0.0123
SPHINCS+ 256s	0.0204	0.0039	0.0997	0.0128	0.051	0.0068	0.2392	0.0129
SPHINCS+ 256f	0.0121	0.0011	0.1169	0.014	0.0777	0.0057	0.2852	0.0133
SNOVA LOW1	0.0732	0.0005	0.0477	0.0028	0.1333	0.0043	0.1393	0.0237
SNOVA MEDIUM1	0.0745	0.0005	0.0805	0.002	0.1362	0.0053	0.1636	0.0315
SNOVA HIGH1	0.0754	0.0007	0.1128	0.0099	0.141	0.0068	0.1602	0.0434
SQISign LOW2	0.0777	0.001	0.0642	0.002	0.1406	0.0069	0.159	0.042
SQISign MEDIUM2	0.0806	0.0007	0.0957	0.004	0.142	0.0056	0.1427	0.0318
SQISign HIGH2	0.0822	0	0.1256	0.004	0.1425	0.0039	0.1747	0.0407
Hawk 256	0.1825	0.028	0.0442	0.0051	0.4781	0.0686	0.1101	0.0077
Hawk 512	0.2205	0.0491	0.0468	0.0053	0.5694	0.124	0.1105	0.0049
Hawk 1024	0.2196	0.057	0.0496	0.0053	0.5603	0.1335	0.1159	0.0111
Dilithium 2aes	0.044	0.0053	0.0497	0.0056	0.1128	0.0052	0.1224	0.005
Dilithium 3aes	0.0426	0.004	0.0436	0.0025	0.1109	0.0019	0.1166	0.0035
Dilithium 5aes	0.048	0.0036	0.0548	0.0084	0.1157	0.0058	0.1298	0.0131

In summary, for each platform and configuration in Table 1, we execute the following workflow: (1) the client submits the primitive identifier, parameterization, and operation type to the server; (2) the server invokes the corresponding compiled implementation either as a standalone executable (*subprocess.run(...)*) or as an in-process library call (DLL via *ctypes*) depending on availability; (3) the wall-clock timing interval is recorded only around the cryptographic invocation boundary defined in Sections 3.2 and 3.4 (excluding TCP send/receive and request parsing/serialization); (4) the operation is repeated ten times and aggregated as mean, standard deviation, and 95% confidence interval; and (5) functional correctness is validated through shared secret agreement for KEMs and signature verification for signature schemes.

3.5. Data Collection and Result Aggregation

For each combination of platform (AMD vs. Intel), primitive (KEM or signature scheme), and operation type, the testbed produces a set of per-run timing values (ten measurements). These values are collected in structured logs and exported to tabular form for post-processing and visualization. The reported results in subsequent sections are derived from these collected measurements, using aggregated indicators (e.g., representative average timings across runs) to compare primitives across platforms under a unified methodology. Figures report mean ±1 standard deviation over 10 runs, and Tables 3 and 4 report the mean and 95% confidence intervals.

3.6. Measured Outputs and Correctness Checks

In addition to timing, the testbed verifies functional correctness for each primitive. For KEMs, the encapsulated and decapsulated shared secrets are checked for agreement. For digital signatures, a generated signature is verified with the corresponding public

key, and the verification outcome is recorded. This ensures that reported performance measurements correspond to successfully executed cryptographic operations rather than failed or degenerate runs.

3.7. Limitations

As with any empirical performance study, the results reported in this paper should be interpreted in light of the following limitations.

Our timings reflect wall-clock execution time as observed by the Python orchestration layer within a TCP client–server testbed. However, by construction, the timed interval starts after the request has been received and inputs are prepared. Therefore, TCP send/receive and request parsing/serialization are excluded from the reported timings. The measured time may still include overhead from the orchestration environment within the timing boundary (e.g., process invocation/start-up for executable-based schemes, wrapper logic, file/I/O used by the invoked implementation, and OS scheduling). This overhead is applied consistently within each integration class, but it may be non-negligible for very fast operations and should be considered when interpreting small differences.

Performance results depend strongly on the specific implementation choices embedded in the invoked executables (e.g., algorithmic optimizations, compiler and build configuration, use of platform-specific instructions, and defensive coding for constant-time behavior). Consequently, the measurements in this work should be understood as evidence for the evaluated implementations as executed in our testbed, rather than as universal lower bounds (or definitive rankings) for the underlying algorithms.

PQC schemes typically offer multiple parameter sets and variants. The performance reported here corresponds to the parameterization configured in the tested executables. Since different parameter sets can shift the time/size trade-offs substantially, our results should not be extrapolated to all configurations without additional measurement.

Experiments were conducted on two commodity processors and under a single operating system family (Windows 11). While these platforms are representative of common endpoints, they do not capture the full diversity of deployment environments (e.g., servers with different microarchitectures, Linux-based infrastructure, mobile devices, microcontrollers, and specialized accelerators). Results may therefore differ on other hardware/software stacks.

Each operation was repeated ten times to reduce transient variability and obtain stable indicators under controlled load. However, this repetition count does not provide comprehensive statistical characterization of variance under diverse system conditions (e.g., background load, thermal throttling, or long-running contention effects). For deployment planning, additional testing under production-like conditions may be required.

The benchmark focuses on primitive-level operations (key generation, encapsulation/decapsulation, signing, verification) in a controlled client–server workflow. It does not directly measure full protocol-level behavior under concurrency (e.g., multi-session TLS termination, PKI renewal at scale) or real wide-area network conditions. Therefore, the results primarily inform primitive selection and relative computational cost, rather than complete end-to-end system throughput in production.

This study emphasizes execution time and includes size-related indicators where available. It does not provide a systematic evaluation of additional deployment-relevant metrics such as memory consumption, energy usage, side-channel resistance, or formal constant-time validation. These factors can be decisive in high-assurance or resource-constrained environments and should be assessed separately when required.

These limitations are common in comparative benchmarking studies and do not detract from the main objective of this work: providing a consistent, platform-level comparison of representative PQC KEMs and signature schemes under a unified measurement procedure.

4. Experimental Results

This section reports the performance measurements obtained with the testbed described in Section 3. Results are presented for both platforms (AMD Ryzen 7 4000 and Intel Core i5-1035G1) and for each evaluated primitive and operation. Unless stated otherwise, all results correspond to the parameterizations configured in the evaluated executables (summarized in Table 1) and the repeated-run procedure described previously.

4.1. Key Generation

Figures 1 and 2 illustrate the average key generation times for KEMs. Across both platforms, Kyber consistently exhibits the lowest key generation latency among the evaluated algorithms. In contrast, the code-based KEMs (HQC and BIKE) show higher key generation times, with the performance gap becoming more pronounced on the Intel platform. This behavior is consistent with the heavier arithmetic and decoding-related operations typically required by code-based constructions.

Regarding digital signatures (see Figures 3 and 4), the measurements indicate meaningful differences across families. Dilithium and HAWK present stable key generation performance across platforms, while SQISign and SNOVA show markedly different computational profiles compared to lattice-based schemes. For SPHINCS+, the results reflect the expected trade-off of hash-based signatures, i.e., key generation behavior depends on the selected variant, and deeper tree configurations may incur higher key generation cost due to the additional internal structure required for stateless signing.

Overall, these measurements suggest that, from the standpoint of key generation cost alone, lattice-based schemes tend to be the most deployment-friendly on commodity CPUs, while alternative families may require more careful engineering to keep latency within acceptable bounds.

4.2. KEM Encapsulation and Decapsulation

Encapsulation (encryption) and decapsulation (decryption) are key operations for network protocols based on KEMs (e.g., TLS key establishment). Figures 5 and 6 show the encryption time for these algorithms on AMD and Intel, respectively, and Figures 7 and 8 show the corresponding decryption times. The results show a consistent ranking across both platforms. On the one hand, Kyber delivers the lowest encapsulation and decapsulation times, indicating strong suitability for high-frequency session establishment and latency-sensitive network services. On the other hand, HQC and BIKE incur higher costs, with the Intel platform exhibiting a stronger penalty relative to AMD in several operations.

The observed differences align with the practical expectations for these families. Lattice-based Module Learning With Errors (MLWE)-style KEMs are typically optimized around structured polynomial arithmetic, whereas code-based KEMs often involve more expensive decoding or syndrome-based operations. From a deployment perspective, the measurements support Kyber as the most practical choice among the tested KEMs when the primary objective is minimizing computational overhead. Table 3 summarizes all the results, showing the mean and the confidence interval.

4.3. Signature Generation

Signature generation cost is particularly important for workloads such as software signing pipelines, authentication services, and high-volume transactional signing. Figures 9 and 10 present the signature generation times on AMD and Intel. The measure-

ments show that Dilithium provides a strong and stable signing performance across both CPUs, maintaining low latency and relatively limited variability. HAWK exhibits signing behavior in the same general range as lattice-based schemes, offering a balanced profile that can be attractive when considering trade-offs beyond raw speed.

By contrast, SQISign and SNOVA exhibit higher signing costs than the lattice-based schemes in our setup, reflecting the more complex computational structure of these non-lattice-based approaches. SPHINCS+ presents a different pattern. Hash-based signatures are appealing for their conservative security foundations, but signing time can increase substantially depending on the chosen parameter set and variant, which may affect suitability for high-throughput signing tasks.

4.4. Signature Verification

Figures 11 and 12 illustrate signature verification times. Verification performance is critical for services that validate large numbers of signatures (e.g., PKI validation, document signing verification, secure update verification on fleets of devices). Our results show that, across platforms, Dilithium again demonstrates efficient and stable verification. HAWK also achieves verification times that remain within a practical envelope for typical applications.

SQISign (isogeny-based) and SNOVA (multivariate) show higher verification latency relative to lattice-based schemes under the evaluated configurations. For SPHINCS+, verification performance depends on the configuration and (consistent with hash-based designs) may remain feasible for verification-centric applications, but it can impose non-trivial overhead at larger parameter sets. Table 4 summarizes all the results, showing the mean and the confidence interval.

4.5. Size Considerations and Deployment Trade-Offs

In addition to timing, deployment feasibility depends on the size of public keys, secret keys, ciphertexts, and signatures. While execution time often dominates for latency-sensitive systems, size overhead directly affects the following: bandwidth consumption during handshakes and authentication exchanges, storage requirements in PKI and key-management systems, and the feasibility of adoption in constrained environments such as IIoT.

As shown in Figures 13 and 14, the evaluated schemes reflect different points in the design space. Lattice-based schemes provide a relatively balanced profile, while hash-based, isogeny-based, and multivariate schemes may offer advantages in specific size dimensions at the cost of increased computation and/or larger artefacts in others. These trade-offs should be evaluated against the target deployment context (e.g., interactive protocols vs. offline signing or server-side termination vs. embedded verification, among others).

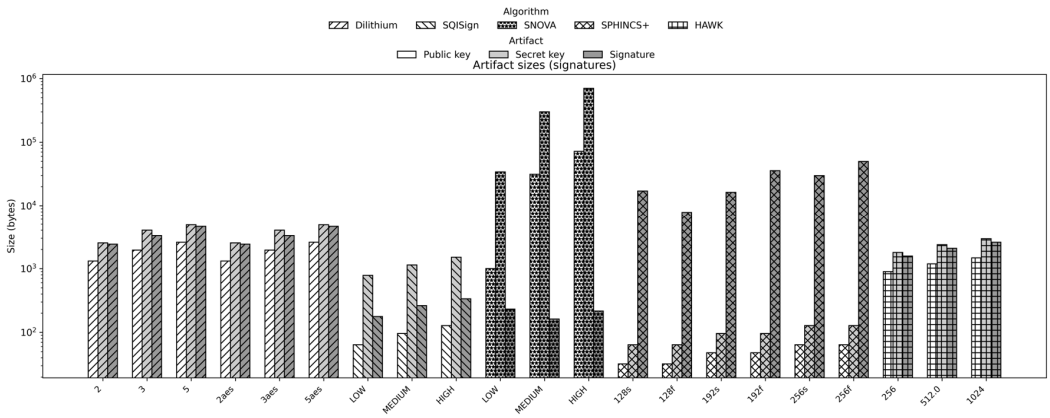

Figure 13. Artifact sizes (bytes and log scale) for the evaluated post-quantum KEM schemes, showing public key, secret key, and ciphertext sizes across the tested parameterizations.

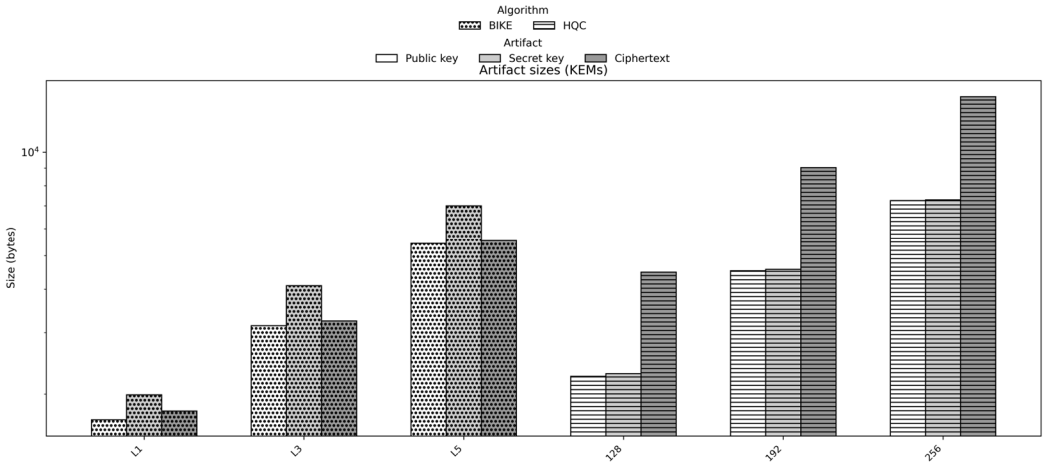

Figure 14. Artifact sizes (bytes and log scale) for the evaluated post-quantum signature schemes, showing public key, secret key, and signature sizes across the tested parameterizations.

4.6. Platform Comparison (AMD vs. Intel)

A consistent observation across all evaluated primitives is that the AMD Ryzen 7 4000 platform outperforms the Intel Core i5-1035G1 platform in most measured operations. While the absolute gap depends on the algorithm and operation type, the trend is stable. PQC primitives, particularly those with heavier arithmetic, benefit substantially from higher parallel capacity and higher base computational throughput.

This platform effect is relevant for transition planning. When estimating PQC operational costs, organizations should consider that the same algorithm can impose materially different overheads depending on the deployed hardware generation and class (endpoint vs. server vs. embedded).

5. Discussion

This section interprets the benchmarking results presented in Section 4, focusing on implications for algorithm selection, implications for common deployment contexts, and limitations and reproducibility considerations.

5.1. Implications for Selecting PQC Primitives

A consistent outcome of our measurements is that lattice-based primitives provide the most favorable efficiency profile on the evaluated commodity CPUs. In particular, Kyber (ML-KEM) exhibits the lowest computational cost among the evaluated KEMs for key generation and encapsulation/decapsulation, making it a strong default candidate for latency-sensitive key establishment. Similarly, Dilithium (ML-DSA) offers stable signing and verification performance across both platforms, supporting its role as a pragmatic baseline for post-quantum authentication in many applications.

By contrast, the code-based KEMs evaluated in this study (HQC and BIKE) incur higher encapsulation/decapsulation times in our setup. From a deployment perspective, this does not reduce their relevance because algorithmic diversity is a recognized risk-management strategy. However, it means that adopting these schemes may require additional engineering effort (optimized implementations or platform tuning) and careful capacity planning, especially on systems where key establishment is performed at high rates.

For digital signatures, the results reinforce the importance of matching the primitive to the operational profile. SPHINCS+ (SLH-DSA) provides strong long-term security assumptions (hash-based) but introduces different cost and size trade-offs, including potentially high signing latency depending on the chosen configuration and relatively large artefacts

compared to lattice-based signatures. This can be acceptable for low-frequency signing use cases, but it may be less attractive for high-throughput transactional signing workloads. On the other hand, SQISign and SNOVA show a higher computational burden under the evaluated configurations, which limits their immediate attractiveness for performance-sensitive systems. However, these schemes remain relevant as research directions and as alternative design points. In some contexts, implementers may accept higher computation if it enables favorable properties elsewhere (e.g., certain size or key-management considerations).

We emphasize that these conclusions are drawn under the end-to-end measurement boundary defined in Section 3.4. Therefore, the results should be interpreted as implementation and integration costs within our testbed, rather than as isolated cryptographic kernel timings. Overall, the results support a practical near-term narrative. ML-KEM/ML-DSA-class schemes are well-positioned for mainstream adoption on general-purpose processors. Other families may be best viewed as either diversity options or specialized candidates requiring stronger justification and optimization work.

5.2. Implications for Deployment Contexts

Implications are addressed in terms of interactive key establishment (e.g., TLS-like handshakes), PKI, identity, and authentication services; software distribution and update ecosystems; and finally constrained and industrial environments (IoT/IIoT).

Regarding key establishment, KEM encapsulation and decapsulation are performed during session establishment and can directly affect latency, throughput, and CPU cost at termination points (e.g., load balancers and application gateways). The relative advantage observed for Kyber suggests that lattice-based KEMs are particularly suitable for high-rate session establishment and environments where key exchange occurs frequently. This is consistent with protocol-level evaluations that measure the overhead of hybrid and post-quantum mechanisms in TLS contexts (e.g., [14,16]), which highlight that the feasibility of PQC in network protocols depends strongly on the computational and communication costs of the selected primitives.

In terms of PKI, identity and authentication, signature verification is often performed at scale (e.g., validation of certificates, signed messages, and software artefacts), whereas signing may be centralized and rate-limited (e.g., certificate issuance, internal code-signing services, etc.). In such settings, schemes with efficient and stable verification, such as Dilithium in our experiments, are attractive because they reduce the cost of widespread verification across endpoints.

Software signing pipelines and firmware update mechanisms typically exhibit an asymmetric workload. A small number of signing operations on the producer side, followed by many verification operations on endpoints. This pattern can accommodate signatures that are computationally heavier for signing (if signing is infrequent), but it may still be sensitive to verification latency and signature size, which impacts bandwidth and storage. Consequently, selecting a signature scheme for these ecosystems requires balancing long-term security objectives with operational constraints (e.g., update package size or verification on heterogeneous endpoints).

The performance trends matter for constrained systems because they indicate how quickly costs can grow as implementations move away from highly optimized primitives. Recent work on PQC overhead in IIoT settings [12] similarly stresses that even moderate overheads can become problematic when CPU, memory, and bandwidth are limited. In these environments, platform-specific optimization and careful protocol design are often prerequisites for deployment.

5.3. Cross-Platform Effects and Capacity Planning

Across all evaluated primitives, the AMD platform consistently outperformed the Intel platform in our measurements. This reinforces a practical lesson for transition planning, which is that PQC performance is not only algorithm-dependent but also hardware-dependent. Organizations estimating the cost of adopting PQC should therefore avoid relying on a single benchmark number. Instead, they should evaluate representative target hardware classes (endpoints, servers, gateways) and consider refresh cycles and scaling strategies. Even when the same primitives are selected, the operational cost profile can vary materially with CPU generation, microarchitecture, and system configuration.

5.4. Relation to Prior Benchmarking Work

Our findings are broadly aligned with the existing benchmarking literature that reports strong performance for NIST-selected lattice-based primitives in practical settings. For instance, Abbasi et al. [13] benchmark Kyber, Dilithium, and BIKE across heterogeneous environments and emphasize the sensitivity of PQC overhead to platform characteristics and deployment assumptions. Protocol context studies, particularly around TLS [14,16], further confirm that practical viability depends on the combined effect of primitive performance, protocol integration, and implementation choices. Work exploring PQC in distributed ledgers similarly illustrates that performance trade-offs may shift when primitives are embedded in complex transaction workflows [17].

Importantly, different studies adopt different measurement boundaries. Many benchmarks focus on in-process microbenchmarks of cryptographic kernels [21], while others evaluate protocol-level or application-level behavior. In contrast, our results reflect an end-to-end primitive invocation boundary through a common TCP client–server orchestration workflow (Section 3.4). Accordingly, cross-scheme rankings should be interpreted cautiously when schemes fall into different integration classes (executable vs. DLL) and are most directly comparable within each class under the same boundary. As a consequence, absolute timings are not directly comparable across studies because they can incorporate additional non-cryptographic overheads such as process invocation, wrapper logic, and artifact handling within the testbed boundary.

In this context, the contribution of our study is not to replace micro benchmarking or protocol-level analysis but to provide a controlled, unified, deployment-proximate comparison across multiple algorithm families—including both standardized baselines and research-stage schemes—on two representative commodity platforms. This supports evidence-based selection and planning decisions, and it highlights the extent to which integration choices can materially affect observed PQC overheads in practical computing environments.

6. Conclusions

This paper presented an experimental performance evaluation of post-quantum cryptographic primitives on commodity general-purpose processors, focusing on three key encapsulation mechanisms (Kyber, HQC, and BIKE) and five digital signature schemes (CRYSTALS-Dilithium, HAWK, SQISign, SNOVA, and SPHINCS+). Using a unified TCP client–server testbed, we measured key generation, encapsulation/decapsulation, signing, and verification times on two representative platforms, namely, AMD Ryzen 7 4000 and Intel Core i5-1035G1, with repeated runs under controlled conditions.

Across the evaluated configurations, the results indicate that lattice-based schemes provide the most favorable performance profile for mainstream deployment on commodity CPUs. Kyber consistently achieved the lowest computational overhead among the evaluated KEMs, and Dilithium exhibited stable and efficient signing and verification per-

formance. The code-based KEMs (HQC and BIKE) and the evaluated signatures SQISign (isogeny-based) and SNOVA (multivariate) incurred higher computational costs in our testbed, illustrating the practical trade-offs that arise when selecting schemes outside the lattice-based family. SPHINCS+, as a hash-based construction, offers conservative security assumptions and a distinct trade-off space that can be appropriate in specific contexts, but its performance characteristics require careful consideration for high-throughput use cases. Finally, the AMD platform consistently outperformed the Intel platform, underlining that hardware characteristics can materially affect observed PQC overheads and should be considered in capacity planning for migration. These findings complement prior PQC benchmarking and protocol integration studies by providing a controlled primitive-level comparison across multiple families, and they can inform practitioners when selecting primitives and estimating operational costs for quantum-resistant deployments.

Future work will extend this benchmark from primitive-level timings to end-to-end evaluation in real web service deployments, capturing operational impact under realistic loads. We will also replicate the experiments on broader hardware platforms and operating systems to improve generalizability. Finally, we will evaluate additional parameter sets and complementary metrics, such as memory or communication overhead, to study the impact of implementation choices on performance.

Author Contributions: Conceptualization, M.-D.C.; methodology, M.-D.C.; software, M.-D.C., J.A.-F. and A.V.-V.; validation, M.-D.C., J.A.-F., A.V.-V. and Y.A.-A.; investigation, M.-D.C., J.A.-F., A.V.-V. and Y.A.-A.; writing—original draft preparation, J.A.-F., A.V.-V. and Y.A.-A.; writing—review and editing, M.-D.C.; visualization, M.-D.C., J.A.-F., A.V.-V. and Y.A.-A.; supervision, M.-D.C.; project administration, M.-D.C.; funding acquisition, M.-D.C. All authors have read and agreed to the published version of the manuscript.

Funding: This work is part of the project R&D&I Lab in Cybersecurity, Privacy, and Secure Communications (TRUST Lab), financed by European Union NextGeneration-EU, the Recovery Plan, Transformation and Resilience, through INCIBE.

Data Availability Statement: The benchmarking orchestration code, raw measurement logs, post-processing scripts, and the figure-generation notebooks/scripts are publicly available in the replication repository referenced in [20].

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Shor, P.W. Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer. *SIAM Rev.* **1999**, *41*, 303–332. [CrossRef]
2. Ali, S.; Wadho, S.A.; Talpur, K.R.; Talpur, B.A.; Alshudukhi, K.S.; Humayun, M.; Talpur, S.R.; Mamun, M.A.A.; Naseem, M.; Abro, A.; et al. Next-Generation Quantum Security: The Impact of Quantum Computing on Cybersecurity—Threats, Mitigations, and Solutions. *Comput. Electr. Eng.* **2025**, *128*, 110649. [CrossRef]
3. Chen, L.; Jordan, S.; Liu, Y.-K.; Moody, D.; Peralta, R.; Perlner, R.; Smith-Tone, D. *Report on Post-Quantum Cryptography*; NIST: Gaithersburg, MD, USA, 2016. [CrossRef]
4. Cherkaoui Dekkaki, K.; Tasic, I.; Cano, M.-D. Exploring Post-Quantum Cryptography: Review and Directions for the Transition Process. *Technologies* **2024**, *12*, 241. [CrossRef]
5. *FIPS 203*; Module-Lattice-Based Key-Encapsulation Mechanism Standard. NIST: Gaithersburg, MD, USA, 2023. [CrossRef]
6. *FIPS 204*; Module-Lattice-Based Digital Signature Standard. NIST: Gaithersburg, MD, USA, 2024. [CrossRef]
7. *FIPS 205*; Stateless Hash-Based Digital Signature Standard. NIST: Gaithersburg, MD, USA, 2023. [CrossRef]
8. Aguilar Melchor, C.; Gaborit, P.; Aragon, N.; Bettaieb, S.; Bidoux, L.; Blazy, O.; Deneuville, J.-C.; Persichetti, E.; Zémor, G.; Bos, J.; et al. HQC. Available online: https://pqc-hqc.org/ (accessed on 14 November 2024).
9. Aragon, N.; Barreto, P.L.; Bettaieb, S.; Bidoux, L.; Blazy, O.; Deneuville, J.-C.; Gaborit, P.; Ghosh, S.; Gueron, S.; Güneysu, T.; et al. BIKE. Available online: https://bikesuite.org/ (accessed on 14 November 2024).

10. Jami, A.R.; Haleem, A. Quantum Computing as an Enabler for Sustainable Circular Economy Implementation in Industry 4.0: A Study. *Hum. Settl. Sustain.* **2025**, *1*, 103–120. [CrossRef]
11. Sood, V.; Chauhan, R.P. Quantum Computing: Impact on Energy Efficiency and Sustainability. *Expert Syst. Appl.* **2024**, *255*, 124401. [CrossRef]
12. Cruz-Piris, L.; Marín-López, A.; Álvarez-Campana, M.; Sanz, M.; Moreno, J.I.; Arroyo, D. Measuring the Impact of Post Quantum Cryptography in Industrial IoT Scenarios. *Internet Things* **2025**, *34*, 101793. [CrossRef]
13. Abbasi, M.; Cardoso, F.; Váz, P.; Silva, J.; Martins, P. A Practical Performance Benchmark of Post-Quantum Cryptography Across Heterogeneous Computing Environments. *Cryptography* **2025**, *9*, 32. [CrossRef]
14. Paquin, C.; Stebila, D.; Tamvada, G. Benchmarking Post-Quantum Cryptography in TLS. In *Proceedings of the Post-Quantum Cryptography, PQCrypto 2020, Paris, France, 15–17 April 2020*; Lecture Notes in Computer Science; Springer: Cham, Switzerland, 2020; Volume 12100, pp. 72–91.
15. Sikeridis, D.; Kampanakis, P.; Devetsikiotis, M. Post-Quantum Authentication in TLS 1.3: A Performance Study. In Proceedings of the 2020 Network and Distributed System Security Symposium, San Diego, CA, USA, 23–26 February 2020; Internet Society: Reston, VA, USA, 2020.
16. Montenegro, J.A.; Rios, R.; Lopez-Cerezo, J. A Performance Evaluation Framework for Post-Quantum TLS. *Future Gener. Comput. Syst.* **2026**, *175*, 108062. [CrossRef]
17. Juaristi, P.; Agudo, I.; Rios, R.; Ricci, L. Benchmarking Post-Quantum Cryptography in Ethereum-Based Blockchains. In Proceedings of the Computer Security, ESORICS 2024 International Workshops, DPM, CBT, and CyberICPS, Bydgoszcz, Poland, 16–20 September 2024; Springer: Cham, Switzerland, 2014; pp. 340–353.
18. Garcia, C.R.; Aguilera, A.C.; Olmos, J.J.V.; Monroy, I.T.; Rommel, S. Quantum-Resistant TLS 1.3: A Hybrid Solution Combining Classical, Quantum and Post-Quantum Cryptography. In Proceedings of the 2023 IEEE 28th International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD), Edinburgh, UK, 6–8 November 2023; IEEE: New York, NY, USA, 2023; pp. 246–251.
19. Souvatzidaki, K.; Limniotis, K. Post-Quantum Key Exchange in TLS 1.3: Further Analysis on Performance of New Cryptographic Standards. *Cryptography* **2025**, *9*, 73. [CrossRef]
20. Algar, J.; Dubetskyy, V.; Villacís, A.; Cano, M.-D. PQC Benchmarking Repo. Available online: https://github.com/trustlabupct/PQCfinal (accessed on 14 December 2025).
21. Carril, X.; Kardaris, C.; Ribes-GonzáLez, J.; Farràs, O.; Hernandez, C.; Kostalabros, V.; González-Jiménez, J.U.; Moretó, M. Hardware Acceleration for High-Volume Operations of CRYSTALS-Kyber and CRYSTALS-Dilithium. *ACM Trans. Reconfig. Technol. Syst.* **2024**, *17*, 41. [CrossRef]

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.