Performance and Storage Analysis of CRYSTALS-Kyber (ML-KEM) as a Post-Quantum Replacement for RSA and ECC

Nicolas Rodriguez Alvarez

IES Parquesol

nicolas.rodalv@educa.jcyl.es

Fernando Rodriguez Merino

Department of Theoretical, Atomic and Optical Physics, University of Valladolid

fernando.rodriguez.merino@uva.es

Abstract

The steady advancement in quantum computer error correction technology has pushed the current record to 48 stable logical qubits, bringing us closer to machines capable of running Shor's algorithm at scales that threaten RSA and ECC cryptography. While the timeline for developing such quantum computers remains uncertain, the cryptographic community must prepare for the transition to quantum-resistant algorithms. CRYSTALS-Kyber, standardized by NIST in 2022, represents a leading post-quantum cryptographic solution, but widespread adoption faces significant challenges. If this migration follows patterns similar to the SHA-1 to SHA-2 transition, organizations may experience prolonged periods of vulnerability, with substantial security and economic consequences. This study evaluates Kyber's practical viability through performance testing across various implementation schemes, utilizing only standard built-in processor acceleration features (AES-NI, ASIMD) without any specialized hardware additions. Our findings demonstrate that Kyber provides robust security guarantees against quantum attacks while maintaining acceptable performance profiles for most contemporary applications, utilizing only commodity hardware with manufacturer-provided acceleration capabilities.

1 Introduction

The theoretical foundations established by Shor and Grover have evolved from academic concepts to practical concerns as quantum computing hardware continues to advance. While experts debate the timeline for achieving fault-tolerant quantum computers capable of running Shor's algorithm at scale, the cryptographic community faces an urgent imperative: the transition to quantum-resistant algorithms cannot wait for quantum computers to become operational. Historical precedents in cryptographic transitions offer sobering lessons about the challenges ahead. The migration from SHA-1 to SHA-2, initiated in 2005 following the discovery of collision vulnerabilities, took over a decade to complete [PS], with many organizations maintaining vulnerable systems well beyond recommended timelines [PS]. This prolonged transition period exposed numerous systems to security risks and highlighted the substantial economic and operational costs associated with delayed cryptographic upgrades. Suppose the transition to post-quantum cryptography follows similar patterns. In that case, organizations may face extended periods of vulnerability to quantum attacks, with potentially catastrophic consequences for digital infrastructure, financial systems, and national security. In response to this challenge, the National Institute of Standards and Technology (NIST) initiated a comprehensive standardization process for post-quantum cryptographic algorithms, culminating in the selection of CRYSTALS-Kyber as the primary Key Encapsulation Mechanism (KEM) standard in 2022. Unlike RSA and ECC, which derive their security from number-theoretic problems vulnerable to quantum attacks, Kyber's security relies on the learning with errors (LWE) problem over module lattices—a mathematical foundation believed to be resistant to both classical and quantum computational attacks. However, theoretical quantum resistance alone does not guarantee practical adoption. The success of any cryptographic standard depends critically on its performance characteristics, storage requirements, and compatibility with existing hardware infrastructure. Previous post-quantum proposals have faced significant barriers to adoption due to excessive computational overhead, prohibitive key sizes, or requirements for specialized hardware acceleration. To address these concerns and evaluate Kyber's practical viability as a replacement for current cryptographic standards, this study conducts a comprehensive performance analysis across multiple architectures and implementation scenarios. This research contributes to the post-quantum cryptography

transition by providing empirical evidence of Kyber's performance characteristics under realistic deployment conditions. Our evaluation methodology utilizes standard hardware acceleration features available in commodity processors (AES-NI, AVX2, ASIMD) without requiring specialized, quantum-resistant hardware additions, ensuring that our findings accurately reflect the performance organizations can expect during real-world deployment. By comparing Kyber's performance against equivalent-security implementations of RSA-7680 and SECP384R1 (ECC) across both x86_64 and ARM64 architectures, we establish benchmarks that inform migration planning and risk assessment for organizations preparing for the post-quantum era.

2 Background

2.1 Classical Cryptographic Schemes

Modern public-key cryptography relies on two primary mathematical foundations that are vulnerable to quantum attacks. **RSA** (Rivest-Shamir-Adleman) cryptography derives its security from the computational intractability of the integer factorization problem—the difficulty of decomposing large composite numbers into their constituent prime factors. This problem becomes exponentially harder as key sizes increase, making RSA-2048 and RSA-4096 computationally infeasible to break with classical computers within reasonable timeframes [LV01] [RSA78]. **Elliptic Curve Cryptography** (ECC) offers an alternative approach based on the elliptic curve discrete logarithm problem (ECDLP) [Mil07]. ECC achieves equivalent security levels to RSA with significantly smaller key sizes—a 256-bit ECC key provides comparable security to a 3072-bit RSA key. The SECP384R1 curve, standardized by the Standards for Efficient Cryptography (SEC), represents a widely deployed ECC implementation offering 192-bit security strength according to NIST guidelines [Bar06]. Both RSA and ECC implementations typically employ hybrid encryption schemes that combine asymmetric and symmetric cryptography. In these systems, a Key Encapsulation Mechanism (KEM) securely exchanges a symmetric key using public-key methods. At the same time, a Data Encapsulation Mechanism (DEM) handles bulk data encryption using faster symmetric algorithms, such as AES or ChaCha20, which are theoretically unbreakable.

2.2 Quantum Algorithms

Shor's algorithm, formulated by Peter Shor in 1994, marked a milestone in quantum computing theory by showing that a sufficiently large, error-free quantum computer could factor large integers efficiently. Whereas the best-known classical algorithms run in subexponential time, Shor's algorithm runs in "polynomial" time, approximately $O(log(N)^3)$ depending on implementation details, making the cryptographic keys based on large-N factorization effectively breakable in negligible time (RSA and ECC) [Sho97]. Meanwhile, Grover's algorithm, introduced by Lov Grover in 1996, provides a quadratic speedup for unstructured search, reducing the classical cost of $O(N)$ to $O(\sqrt{N})$. Although not initially intended for factorization, Grover's amplitude-amplification technique can be used to optimize specific subroutines within Shor's method or to accelerate searches among partial solutions generated by Shor's quantum circuit [Gro96]. In theory, combining Shor's and Grover's algorithms could optimize the number of iterations and resource usage.

2.3 "Standard" hardware accelerations

In this study, the default hardware accelerations provided by the CPU were utilized. The following section provides a comprehensive explanation of these accelerations.

2.3.1 Intel

Intel® Advanced Encryption Standard Instructions (**AES-NI**): This hardware acceleration provides a speedup of 3 to 10x over an entirely software implementation using AES [Int].

Intel® Advanced Vector Extensions (**AVX/AVX2**): Intel's vector instruction set for SIMD vector operations [Cor21]. The Kyber implementation used in this study leverages AVX2 instructions to accelerate its core lattice-based computations, resulting in significant performance improvements.

Intel® Secure Key (**RDRAND/RDSEED**): On-chip, NIST-certified random number generator instructions for high-quality entropy source in key generation [Cor21]

Intel® Carry-Less Multiplication (**PCLMULQDQ**): Provides a single-cycle, hardware-accelerated carry-less 64x64-bit multiply, used in GCM and other Galois-field operations [Cor21].

2.3.2 ARM

AES: ARM v8 Cryptography Extensions add AESE/AESD/AESMC/AESIMC instructions for single-round encryption/decryption and key-schedule support [ARM14].

SHA1/SHA2: SHA1C / SHAP / SHAM / SHA256H / SHA256SU instructions accelerate SHA-1 and SHA-224/256 hashing [ARM14].

Polynomial Multiply (**PMULL**): 64x64-bit carry-less multiply for efficient GCM-mode Galois-field operations [ARM14].

Advanced SIMD (**ASIMD**): The "NEON" AArch64 SIMD unit for 128-bit vector arithmetic, logical, and data-rearrangement operations [ARM14].

Half-Precision SIMD (**ASIMDHP**): Extension enabling SIMD operations on 16-bit floating-point data types [ARM14].

3 Cryptographic Schemes

To ensure a fair comparison, the following widely known and commonly used algorithms will be employed: RSA-7680, ECDH-SECP384R1 (ECC), and ML-KEM768 (Kyber). They all have the same security strengths according to the NIST [Bar06].

For a practical and fair performance evaluation, it is crucial to test these algorithms as they would be used in a real-world application. Asymmetric cryptography is typically used not for bulk data encryption, but to secure a shared symmetric key or shared secret. This is known as hybrid encryption [CS01], this was implemented according to the NIST recommendations [BCD20].

Therefore, this study evaluates each algorithm within a **hybrid encryption scheme**, which combines a Key Encapsulation Mechanism (KEM) for the asymmetric part, and a Data Encapsulation Mechanism (DEM) for the symmetric part.

To ensure consistency, the same DEM was used for all three schemes: the **ChaCha20-Poly1305** authenticated encryption with associated data (AEAD) cipher [NL18]

A key differentiator between these schemes lies in the mathematical problems that underpin their security. RSA's security relies on the presumed difficulty of the **integer factorization problem** [RSA78], while ECC's is based on the **elliptic curve discrete logarithm problem (ECDLP)** [Kob87] [Mil07]. Both of these problems are known to be efficiently solvable by a sufficiently large quantum computer using Shor's algorithm [Gro96].

In contrast, CRYSTALS-Kyber's security is based on the hardness of solving the **learning with errors (LWE) problem** over module lattices. The LWE problem is widely believed to be resistant to attacks from both classical and quantum computers, which forms the foundation of its post-quantum security claims [Nat24].

4 Benchmarking Methodology

This section outlines the environment and procedures used for evaluating the performance of the cryptographic schemes.

4.1 Hardware

To ensure the veracity of the results, the two most popular architectures were tested on two separate systems: one using the ARM64 architecture—commonly found in portable devices such as the iPhone, Steam Deck, Apple Silicon Macs, and the Raspberry Pi—and the other using the x86_64 architecture, which is prevalent in servers and desktop computers built with Intel® or AMD® processors. This comparison does not consider hardware-accelerated implementations of certain cryptographic operations, such as RSA ones, which can significantly improve performance [Sha05]. The standard ones that the CPU manufacturer implements have been enabled.

Table 1: Key Hardware Specifications of x86_64 Test System

Feature	**Specification**
Architecture	x86_64 (64-bit mode)
CPU Vendor	Intel®
Processor Model	Xeon E5-2686 v4
Base Frequency	2.30 GHz
CPU Cores	1
Threads per Core	1
Cache Hierarchy	
L1 Data Cache	32 KiB
L1 Instruction Cache	32 KiB
L2 Cache	256 KiB
L3 Cache	45 MiB
Cryptography-Relevant Instruction Sets	
AES-NI	Supported
AVX/AVX2	Supported
PCLMULQDQ	Supported
RDRAND	Supported

Table 2: Key Hardware Specifications of ARM64 Test System

Feature	**Specification**
Architecture	ARM aarch64 (64-bit)
CPU Vendor	ARM
Processor Model	Neoverse-N1
CPU Cores	2
Threads per Core	1
Stepping	r3p1
Cache Hierarchy	
L1 Data Cache	128 KiB (64 KiB/core)
L1 Instruction Cache	128 KiB (64 KiB/core)
L2 Cache	2 MiB (1 MiB/core)
L3 Cache	32 MiB (shared)
Cryptography-Relevant Instruction Sets	
AES	Supported
SHA1/SHA2	Supported
PMULL (Polynomial Multiply)	Supported
ASIMD (Advanced SIMD)	Supported
ASIMDHP (FP16 support)	Supported

The benchmarking environment employs two distinct systems representing prevalent computing architectures: an x86_64 platform (Intel® Xeon E5-2686 v4) and an ARM64 platform (ARM Neoverse-N1). Both configurations deliberately emulate general-purpose CPU scenarios by disabling hardware acceleration for asymmetric cryptographic operations (e.g., RSA modular exponentiation

units), ensuring fair algorithmic comparisons under standardized software implementations. The x86_64 system reflects server/desktop profiles with a single-core setup (2.3 GHz base frequency) and a large 45 MiB shared L3 cache, while the ARM64 system mirrors edge/IoT constraints with a dual-core design and 32 MiB shared L3 cache. Crucially, both platforms support modern cryptographic instruction sets: x86_64 leverages AES-NI, AVX/AVX2, and PCLMULQDQ, whereas ARM64 utilizes AES, PMULL, and ASIMD extensions. For these benchmarks, all available accelerations, including AVX2 for the Kyber implementation, were used where applicable by the cryptographic libraries. These instructions were fully enabled during testing to reflect real-world deployment conditions. The absence of **dedicated** asymmetric hardware acceleration ensures results reflect baseline CPU performance relevant to widespread software deployments, with cache hierarchies (L1-L3) and single-threaded execution isolating per-core computational bottlenecks inherent to cryptographic workloads.

4.2 Benchmarking Software

The two systems have utilized the Ubuntu Linux distribution, specifically the 24.04.2 LTS version. On the programming side, Rust was used as the programming language for the benchmarks. The openssl crate provided the implementations of RSA and SECP384R1, while the oqs crate, which enables the use of ML-KEM via the Rust bindings for the Open Quantum Safe's liboqs library.

The benchmarks are consolidated into two main sections: Performance and Storage. The Performance section measures the necessary computational resources required to execute a specific operation, measured in CPU cycles obtained through the use of the iai-callgrind crate. It is essential to note that the CPU cycles are an approximation due to the noise generated by the CPU boost; however, they closely approximate the actual value with a high degree of accuracy. On the other hand, the Storage section measures the size of the outputs of the ciphers, with a given message. **The complete benchmarking source code is publicly available on this repo nichokas/kyber-performance.** Implementations leverage platform-supported hardware acceleration (AES-NI, PMULL, ASIMD) for relevant operations. Asymmetric-specific hardware (e.g., RSA modular exponentiation units) was turned off to ensure algorithmic fairness.

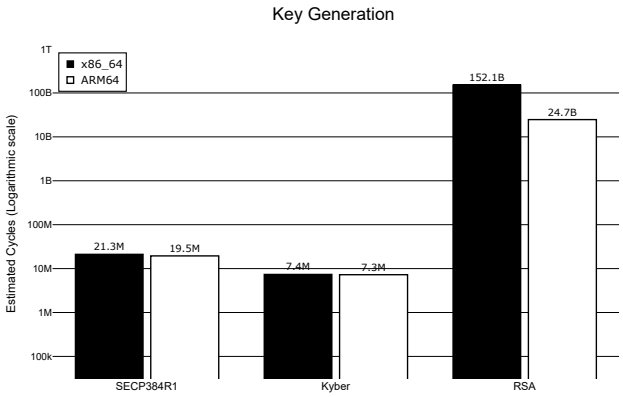
5 Comparison

For reproducibility, the version of the source code used for this paper is Commit 6fa6b0c.

5.1 Speed benchmarks

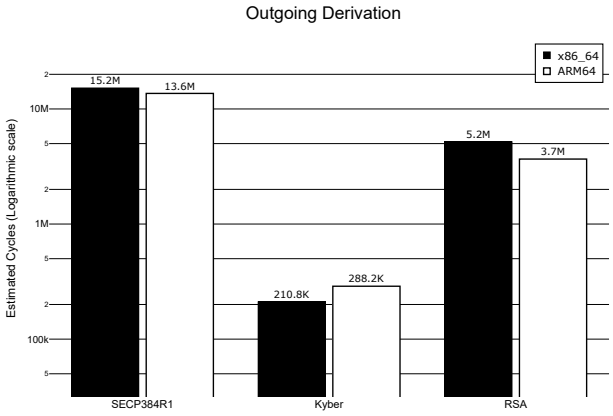
5.1.1 Key Generation

Measurement of the required computational resources to create a new public-private keypair.



Kyber demonstrates superior efficiency in key generation. On both x86_64 and ARM64 architectures, Kyber requires the fewest cycles (7.4M and 7.3M, respectively), making it approximately 2.7 to 3 times faster than SECP384R1 (21.3M and 19.5M cycles). The performance gap with RSA is staggering. On the x86_64 platform, Kyber is over 20,500 times faster than RSA, and on ARM64, it remains over 3,400 times faster. This vast difference is due to the underlying mathematics of the algorithms. RSA key generation relies on the computationally intensive and time-consuming process of finding large prime numbers, which scales poorly. In contrast, Kyber's lattice-based arithmetic allows for much more efficient key creation. Although the ARM64 architecture significantly reduces RSA's key generation time by a factor of six compared to x86_64, it remains orders of magnitude slower than both Kyber and ECC.

5.1.2 Outgoing Shared Secret Derivation

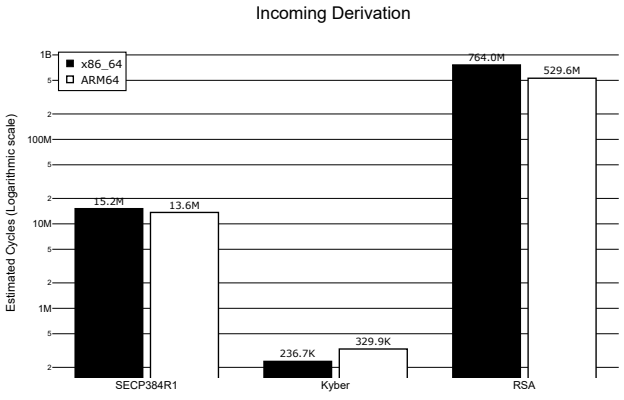


This chart illustrates the performance of deriving a shared secret from the initiator's (the "outgoing" party's) perspective. This process typically involves using one's own private key and the recipient's public key to establish a mutual secret. The y-axis is on a logarithmic scale to properly visualize the performance differences in estimated CPU cycles.

The data reveals that Kyber is the most efficient algorithm for this operation on both tested platforms. On x86_64, Kyber's cost of approximately 210,800 cycles is roughly 25 times faster than RSA (5.2 million cycles) and 72 times faster than SECP384R1 (15.2 million cycles). On the ARM64 architecture, Kyber (288,200 cycles) maintains its lead, proving to be about 13 times faster than RSA (3.7 million cycles) and 47 times faster than SECP384R1 (13.6 million cycles).

It is important to note that for RSA, this "public-key" operation is significantly less costly than the "private-key" operation shown in the "Incoming Derivation" chart, which is an expected characteristic of the algorithm. Nevertheless, Kyber still surpasses it by a substantial margin. For SECP384R1, the computational workload is identical for both the incoming and outgoing phases, which explains its consistently high cycle count. Kyber's efficiency, requiring fewer than 300,000 cycles, facilitates near-instantaneous symmetric key establishment, drastically reducing the latency of cryptographic handshakes compared to both RSA and ECC.

5.1.3 Incoming Shared Secret Derivation



On the x86_64 architecture, Kyber requires only about 236,700 cycles. This is approximately 64 times more efficient than SECP384R1, which takes 15.2 million cycles, and over 3,200 times more efficient than RSA, which needs 764 million cycles for the same task. A similar trend is observed on the ARM64 platform, where Kyber's 329,900 cycles outperform SECP384R1 (13.6 million cycles) by a factor of 41 and RSA (529.6 million cycles) by a factor of over 1,600.

It is important to note that for

SECP384R1, the derivation procedure is identical for both the outgoing and incoming phases, which explains its consistently high computational cost compared to Kyber in both scenarios. This gap stems from Kyber's foundation in module lattice-based arithmetic, which avoids the expensive elliptic curve point multiplication used by SECP384R1 and the even more costly modular exponentiation of RSA.

While the ARM64 architecture reduces the absolute cycle count for classical algorithms, it does not alter the fundamental performance hierarchy.

5.2 Storage benchmarks

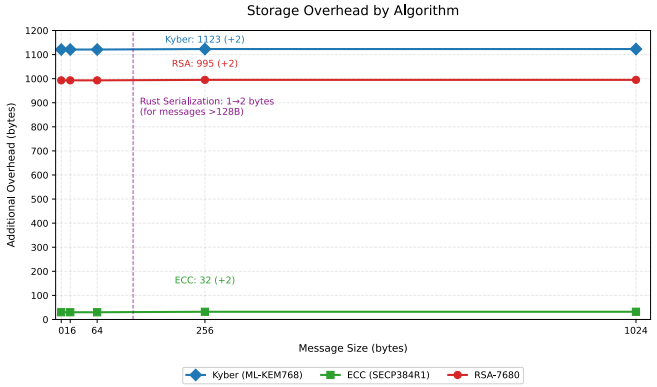


Table 3: Added overhead (without the rust bytes)

	nonce	Key transport	Total added length
SECP384R1	30		30
Kyber	30	1091	1121
RSA	30	963	993

The added overhead refers to the data that an algorithm adds to the existing data (plaintext length) to encrypt it, and in some cases, to include the key transport, for example, with Kyber, is: plaintext message length + Kyber's overhead + Rust's serialization bytes. The storage overhead analysis reveals a critical trade-off between quantum resistance and bandwidth efficiency. While the Poly1305 authentication nonce contributes a constant 30-byte overhead across all schemes, the key transport mechanism diverges significantly: SECP384R1 (ECC) requires zero key transport overhead due to its elliptic curve key agreement protocol, resulting in a minimal total added length of 30 bytes. In contrast, Kyber and RSA introduce substantial key transport payloads (1,091 bytes and 963 bytes, respectively), resulting in total overheads of 1,121 bytes for Kyber and 993 bytes for RSA. This establishes ECC as the most bandwidth-efficient scheme—advantageous in constrained environments, such as IoT—but its quantum

vulnerability remains an existential risk. Kyber's 13.5% higher transport overhead than RSA is justified by its quantum resistance and superior computational efficiency (see Section 5.1), positioning it as the optimal choice for future-proof systems where bandwidth permits.

Conclusion

The benchmarking results demonstrate CRYSTALS-Kyber's decisive computational advantages over classical cryptographic schemes across both `x86_64` and `ARM64` architectures. These gains reflect Kyber's natural alignment with modern CPU features; its implementation leverages vectorization instructions such as **AVX2** to accelerate core lattice-based computations. Kyber operates **2.7–3× faster than ECC** and **3,400×–20,500× faster than RSA** in key generation due to its efficient lattice-based polynomial arithmetic. For shared secret derivation, Kyber maintains **41–72× speed advantages over ECC** and **1,600×–3,200× advantages over RSA**, effectively eliminating RSA's critical bottleneck in receiver-side operations. These performance gains are consistent across architectures, with `ARM64` partially mitigating the overheads of classical schemes without altering the fundamental performance hierarchy. The sole trade-off emerges in storage requirements, where Kyber's 1,121-byte hybrid payload exceeds RSA's 993 bytes and ECC's minimal 30-byte overhead, reflecting the inherent tension between quantum resistance and bandwidth efficiency. Collectively, these results establish Kyber as delivering NIST-standardized quantum resistance without computational penalties.

These highly positive results, obtained using built-in CPU features like AVX2 and AES-NI, strongly suggest a path for future innovation: the design and development of specialized hardware accelerators for lattice-based cryptography. The creation of such hardware could unlock further performance gains, reducing latency and power consumption to solidify the viability of post-quantum cryptography in even the most resource-constrained environments, such as IoT devices.

Acknowledgements

I would like to extend my sincere gratitude to Elías F. Combarro for his invaluable mentorship on the intricacies of academic writing and for the constructive feedback he provided during the development of this manuscript. His support was instrumental in shaping this work.

References

[ARM14] ARM Limited. ARM® Cortex®-A53 MP-Core Processor Cryptography Extension Technical Reference Manual (DDI 0501F). Technical Reference Manual DDI 0501F, ID041316, ARM Limited, 2014. Optional Cryptography Extension for AES and SHA on Cortex-A53.

[Bar06] Elaine Barker. Suite B Cryptography, March 2006. (accessed April 26, 2025).

[BCD20] Elaine Barker, Lily Chen, and Richard Davis. Recommendation for key-derivation methods in key-establishment schemes. Technical report, National Institute of Standards and Technology, August 2020.

[Cor21] Intel Corporation. *Intel 64 and IA-32 Architectures Software Developer's Manual Volume 2: Instruction Set Reference*. Intel Corporation, September 2021.

[CS01] Ronald Cramer and Victor Shoup. Design and analysis of practical public-key encryption schemes secure against adaptive chosen ciphertext attack. Cryptology ePrint Archive, Paper 2001/108, 2001.

[Gro96] Lov K. Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*, STOC '96, page 212–219, New York, NY, USA, 1996. Association for Computing Machinery.

[Int] Intel Corporation. Intel® Advanced Encryption Standard (Intel® AES) Instructions Set, Rev 3.01. Online. Last updated August 2, 2012.

[Kob87] Neal Koblitz. Elliptic curve cryptosystems. *Math. Comput.*, 48(177):203–209, 1987.

[LV01] Arjen K. Lenstra and Eric R. Verheul. Selecting cryptographic key sizes. *Journal of Cryptology*, 14(4):255–293, 2001.

[Mil07] Victor S Miller. Use of elliptic curves in cryptography. In *Lecture Notes in Computer Science*, pages 417–426. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007.

[Nat24] National Institute of Standards and Technology. FIPS 203: Module-Lattice-Based Key-Encapsulation Mechanism Standard. FIPS Publication 203, National Institute of Standards and Technology, Information Technology Laboratory, Gaithersburg, MD, USA, August 2024.

[NL18] Yoav Nir and Adam Langley. ChaCha20 and Poly1305 for IETF Protocols. RFC 8439, June 2018.

[PS] Chris Palmer and Ryan Sleevi. Gradually sunsetting SHA-1. Online.

[RSA78] R L Rivest, A Shamir, and L Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, February 1978.

[Sha05] Ajay C. Shantilal. A faster hardware implementation of rsa algorithm. Technical report, Department of Electrical & Computer Engineering, Oregon State University, Corvallis, Oregon, USA, 2005.

[Sho97] Peter W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Journal on Computing*, 26(5):1484–1509, October 1997.

6 Raw Obtained Data

Table 4: Consolidated Cryptographic Performance Metrics (x86_64 & ARM64)

Algorithm	Operation	Arch	Instr	L1 Hits	L2 Hits	RAM Hits	Est. Cycles
SECP384R1	Incoming Secret	x86_64	11658237	15117387	4505	1912	15206832
SECP384R1	Incoming Secret	ARM64	10231635	13557483	4534	1859	13645218
SECP384R1	Outgoing Secret	x86_64	11658237	15117387	4505	1912	15206832
SECP384R1	Outgoing Secret	ARM64	10231635	13557483	4534	1859	13645218
SECP384R1	Key Generation	x86_64	15975363	20939189	12050	9278	21324169
SECP384R1	Key Generation	ARM64	14259264	19082149	12224	9094	19461559
Kyber	Incoming Secret	x86_64	177064	212675	2211	370	236680
Kyber	Incoming Secret	ARM64	268713	319023	413	251	329873
Kyber	Outgoing Secret	x86_64	152513	184443	2332	419	210768
Kyber	Outgoing Secret	ARM64	229406	272873	833	318	288168
Kyber	Key Generation	x86_64	5266214	7050285	10400	9543	7436290
Kyber	Key Generation	ARM64	5082726	6893944	10393	8955	7259334
RSA	Incoming Secret	x86_64	609884957	763988457	4563	665	764034547
RSA	Incoming Secret	ARM64	459230402	528209769	273797	494	529596044
RSA	Outgoing Secret	x86_64	4229026	5169943	1523	335	5189283
RSA	Outgoing Secret	ARM64	3124179	3652723	1507	266	3669568
RSA	Key Generation	x86_64	121603553478	152134665100	942996	20601	152140101115
RSA	Key Generation	ARM64	21637354719	24672791454	10775986	20933	24727404039