International Journal of Information Security manuscript No.
(will be inserted by the editor)

Post-Quantum Cryptography: A Multi-layer Bottleneck Analysis

Ruben Rios · José A. Montenegro

Received: date / Accepted: date

Abstract Advances in quantum computing have led the research community to develop new post-quantum cryptographic algorithms. While most efforts have focused on designing and analyzing the security of these algorithms, their integration into real-world systems demands a multi-layer performance analysis to identify and mitigate potential bottlenecks. Aligned with NIST's post-quantum transition strategy, this work addresses that need by introducing a performance evaluation framework that enables to assess and analyze the impact of post-quantum key encapsulation mechanisms (KEMs) and signatures at multiple layers of the protocol stack—from standalone cryptographic primitives, to their integration into the TLS handshake, and finally into full application scenarios such as VPNs—on two hardware platforms. Our results show that signature and verification operations dominate the overall cryptographic cost, while KEMs such as `MLKEM` introduce minimal overhead. In contrast, hybrid and `HQC`-based solutions significantly increase handshake duration, particularly on ARM platforms. Notably, the relative performance between platforms shifts when comparing these cryptographic primitives in isolation versus their integration into higher protocol layers. This highlights the critical need to evaluate post-quantum algorithms in realistic, system-level scenarios.

Ruben Rios
Network, Information and Computer Security (NICS) lab
Universidad de Malaga, Malaga, 29071, Spain
E-mail: ruben.rdp@uma.es

José A. Montenegro
Network, Information and Computer Security (NICS) lab
Universidad de Malaga, Malaga, 29071, Spain
E-mail: jmmontes@uma.es

Keywords Post-Quantum Cryptography (PQC) · Key Encapsulation Mechanisms (KEM) · Transport Layer Security (TLS) · Virtual Private Networks (VPN) · Performance Benchmarking

1 Introduction

The rapid progress in quantum computing poses a significant threat to traditional cryptographic algorithms [1], such as elliptic curve cryptography (ECC) and RSA, sparking a global effort to develop and standardize post-quantum cryptographic (PQC) algorithms capable to resist quantum attacks. A leading role in this effort was taken by U.S. National Institute of Standards and Technology (NIST) who initiated a competition to standardize PQC algorithms [2], culminating in several FIPS standards for key encapsulation (KEM) and digital signatures.

While the primary focus of the research community has been on the security guarantees of these new algorithms, evaluating the impact of their deployment in real-world systems is critical to a post-quantum transition strategy [3]. Modern systems and applications will heavily rely on secure communication protocols, and the integration of PQC algorithms are not sufficiently explored and may significantly affect latency, scalability, bandwidth, resource consumption and even protocol operation [4].

Effective integration of PQC into existing protocols, such as Transport Layer Security (TLS), and applications, such as Virtual Private Networks (VPNs), requires a thorough understanding of performance tradeoffs across different layers of the communication stack and across various hardware platforms. Consequently, in this work, we present a comprehensive performance

evaluation framework tailored for the post-quantum transition. We systematically assess the performance of post-quantum KEMs and digital signatures—both in isolation and within different layers of the protocol stack—on two representative hardware platforms. Our evaluation allows exploration of architectural differences and their influence on cryptographic performance, providing insights into potential bottlenecks.

Our analysis spans from low-level cryptographic primitive benchmarks to evaluations in TLS handshakes and TLS-based VPN deployments, revealing that signature and verification operations dominate cryptographic overhead, while certain post-quantum KEMs (e.g., `ML-KEM`) introduce minimal latency. In contrast, hybrid and `HQC`-based solutions significantly increase handshake duration, particularly on ARM platforms. This multi-layer evaluation demonstrate that performance rankings can shift dramatically when primitives are embedded in real-world contexts, with some signature schemes introducing notable overheads. These findings underscore the critical need for system-level evaluations to guide the practical adoption of post-quantum cryptography.

The structure of this paper is organized as follows. Sec. 2 analyzes related work and positions our main contributions. Sec. 3 presents our evaluation setup, including the selection of cryptographic primitives, protocols, and applications selected for evaluation, as well as the details of evaluation framework for performance assessment at different layers of the communication stack. Sec. 4, 5 and Sec. 6 present our performance analysis at the primitive, TLS handshake, and VPN application levels, respectively. Sec. 7 discusses the key cases in our work. Finally, Sec. 8 concludes with key findings and outlines future research directions.

2 Related Work

The development of PQC algorithms has been an active area of research, particularly after the announcement of the NIST competition in 2016 [2]. Several works have focused on the analysis of these algorithms from a security perspective (e.g., [5, 6]) while others have concentrated on their applicability and cost.

Some authors have focused on benchmarking different aspects of PQC algorithms. For example, an extensive compilation of performance metrics across various families of algorithms is presented in [7]. Other works have analyzed the energy consumption of these algorithms [8]. In particular, some studies have examined their suitability for power-constrained scenarios, such as the Internet of Things [9]. Similarly, other authors have worked on the optimization and benchmarking of PQC algorithms. For instance, Dang et al. [10] implemented three lattice-based KEMs from the second round of the NIST PQC standardization process, showing that hardware acceleration can substantially improve their execution times. In [11], two competition finalists—one KEM and one signature scheme—are benchmarked on a cryptoprocessor optimized for their execution.

The integration of post-quantum KEMs and digital signatures into security protocols has also been extensively studied. Notably, [12] evaluates NIST signature algorithm candidates at that time and investigates the latency they impose on TLS 1.3 connection establishment. [13] goes one step further and analyzes the impact of both several post-quantum key encapsulation and signature algorithms in TLS. In [14], the authors devise a modular framework to evaluate the impact of integrating post-quantum and hybrid cryptographic primitives in the TLS protocol under various network conditions. Similarly, [15] presents an evaluation across three security levels, based on the traditional and post-quantum candidate algorithms available at the time of the study. Their work includes a breakdown of library-level computational workload, highlighting an increased CPU usage at the protocol level, but without a detailed analysis of the underlying cryptographic primitives. Interestingly, Farooq et al. [16] observed that some KEMs in TLS implementations— particularly `BIKE` and `Classic McEliece`—performed better on Linux platforms than on Windows on x86 systems. Additionally, the authors in [17] focus on analyzing vectorized instruction sets on x86 platforms, demonstrating performance improvements over non-vectorized implementations. However, these improvements are not reflected when integrated into the TLS protocol. That work conducts a two-level analysis, whereas our study performs a three-level analysis across two different platforms.

Some works have also investigated the impact of integrating PQC algorithms in virtual private network applications. [18] evaluated the number of CPU instructions of several post-quantum cryptography from the NIST competition in OpenVPN and HTTPS. Similarly, [19] compared execution time of OpenVPN with traditional `RSA` and with `Dilithium`. Paquin et al. [20] have also worked on the integration of PQC algorithms into the OpenVPN protocol. Some authors have focused on different types of VPN. For example, Bae et al. [21] evaluate the execution speed of PQC algorithms from NIST Round 3 finalists into IPSec. Wireguard VPNs have also received attention lately [22, 23].

The analysis of the literature reveals that the performance of PQC algorithms is influenced by hardware platform; however, most works tend to focus on a single platform. In general, research also tends to concentrate

on a single layer of the protocol stack and does not trace the impact of these algorithms across multiple layers. In contrast, our analysis is not only multi-layered but also considers two widely used hardware architectures. Additionally, we focus on standardized PQC algorithms or those recently selected for standardization. Compared to our previous work [14], this study extends the analysis by incorporating performance comparisons across CPU architectures and evaluating the latest PQC standards. Most notably, it introduces an additional layer of complexity to the assessment of PQC impact on network protocols, namely the VPN layer.

3 Evaluation setup

This section first presents the selection of cryptographic primitives—traditional, hybrid and post-quantum— protocols and tools to be evaluated. Next, it describes the software platform devised for the evaluation of the selected primitives, protocols and tools.

3.1 Primitives, protocols and applications

Cryptographic primitives are typically categorized based on the level of security they provide, which reflects the computational effort required to break them. In traditional cryptography, this is expressed in terms of bits of security, whereas post-quantum cryptography uses security levels—from Level I to V. Although there is no exact correspondence between these two metrics, they are conceptually aligned: for instance, a Level I post-quantum cryptosystem offers resistance to quantum attacks comparable to the resistance of AES-128 against classical attacks. Similarly, Levels III and V correspond approximately to the security offered by AES-192 and AES-256, respectively.

Table 1 presents the cryptographic primitives selected for evaluation. A baseline set of primitives for both key exchange and authentication has been chosen. Schemes based on elliptic curves are prioritized over finite-field alternatives due to their superior efficiency [24]. For post-quantum cryptography, we selected efficient algorithms that have been standardized or selected for standardization in the NIST competition [2]. Hybrid algorithms combining traditional and post-quantum primitives were considered for key encapsulation, but not for digital signatures. This reason for this is that hybrid schemes are primarily designed to mitigate "harvest now, decrypt later" attacks, which do not pose a significant threat to signature schemes.

These cryptographic primitives are commonly used in security protocols to provide authentication and to ensure confidentiality and integrity following key exchange mechanisms. Among the various protocols that incorporate key exchange and authentication, we select the Transport Layer Security (TLS) protocol due to its widespread adoption and versatility across diverse application scenarios. TLS functionality is implemented by various cryptographic libraries, each offering different levels of performance, configurability, and support for algorithms. For our evaluation, we select OpenSSL, an actively maintained open-source library that supports the integration of external cryptographic providers, enabling the use of both traditional and post-quantum cryptographic primitives.

Table 1: Cryptographic primitives selected for evaluation.

Security	KEM			Signature	
Level	Trad	Hybrid	Post	Trad	Post
I	`X25519`	`x25519_mlkem512` `x25519_hqc128`	`mlkem512` `hqc128`	`Ed25519`	`mldsa44`
III	`X448`	`x448_mlkem768` `x448_hqc192`	`mlkem768` `hqc192`	`secp384r1`	`mldsa65`
V	`P-521`	`p521_mlkem1024` `p521_hqc256`	`mlkem1024` `hqc256`	`secp521r1`	`mldsa87`

Several applications leverage the TLS protocol to provide security at higher layers of the protocol stack. Many application-layer protocols—such as HTTP, SMTP, and FTP—rely on TLS to secure their communication channels and ensure confidentiality, integrity, and authentication. Among the various use cases, we select Virtual Private Networks (VPNs) as the target application for testing. In particular, we choose OpenVPN, a widely used open-source VPN solution that employs TLS for secure key exchange and authentication, making it suitable for our evaluation.

3.2 Evaluation Framework

A systematic evaluation of post-quantum cryptography bottlenecks demands for a benchmarking platform that facilitates the configuration, deployment, execution and analysis of cryptographic primitives, protocols and applications.

Fig. 1 presents a layered structure of the tools and libraries that compose the framework, where each layer fulfills a specific function. These components are encapsulated within a containerized environment to ensure consistent deployment of both client and server systems, regardless of the platform architecture. By leveraging containerization, the framework maintains a standardized environment across all experimental setups. For this purpose, `Docker` has been adopted as the core container technology.

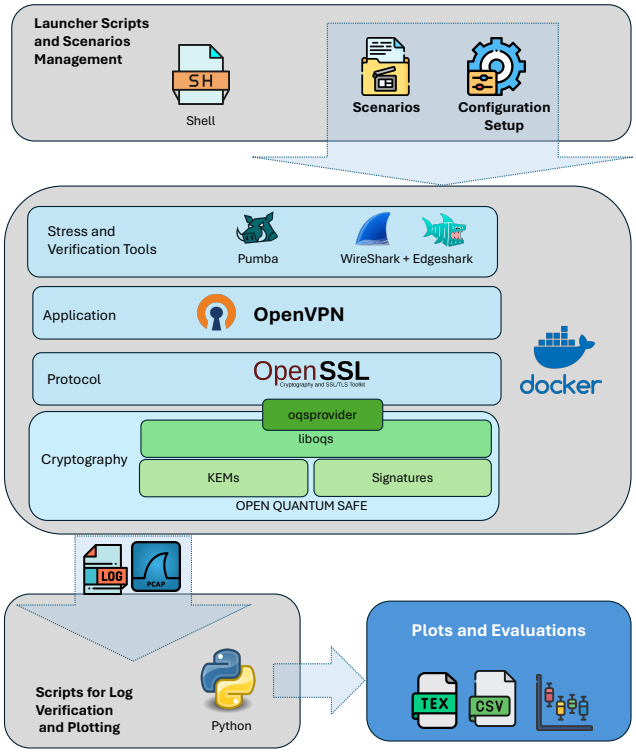


Fig. 1: Evaluation Framework Components

Cryptography layer: this layer provides the primitives (signatures and KEMs) necessary for the protocols to perform the cryptographic handshake. While both `OpenSSL` and `MsQuic` include traditional signature and KEM primitives, they do not include post-quantum primitives by default. This layer relies on the `liboqs` library, from the Open Quantum Safe (OQS) Project, which provides numerous post-quantum cryptographic implementations, including algorithms standardized by NIST as well as proposals from previous standardization rounds.

The integration of these post-quantum primitives into both `OpenSSL` and `MsQuic` is facilitated by their modular architecture, which supports the use of cryptographic providers. The `oqsprovider` acts as bridging component, allowing both protocol implementations to access the primitives offered by `liboqs`.

Protocols Layer: this layer comprises the protocols under evaluation, in this case, TLS. Specifically, `OpenSSL` has been selected due to its widespread adoption and robust support for cryptographic standards. It provides implementations for both client and server components, enabling precise measurement of handshake performance with minimal development effort.

Application Layer: this layer hosts the applications that rely on the TLS protocol implemented in the Protocols layer. In this work, we selected *OpenVPN* as the representative application for evaluation. However, the specific choice of application is not critical, as long as it employs TLS-based authentication to establish a secure communication channel. The primary objective at this layer is to enable a comparison of the TLS handshake duration when executed directly at the protocol level versus during the application runtime, with the aim of analyzing how the handshake time is distributed across both contexts.

Network Layer: this layer provides the capability to emulate various network conditions between the client and server. This makes it possible to evaluate handshake performance in both ideal and realistic scenarios, including packet loss and latency.

Emulating ideal conditions allows for isolating the performance of cryptographic primitives during the handshake, eliminating external factors that could obscure the analysis. Nonetheless, it is equally important to assess performance under realistic network conditions. To this end, our framework integrates `Pumba`, a flexible and powerful tool for manipulating network behavior at the container level.

In addition, the network layer incorporates `EdgeShark`, which enables `Wireshark` to capture packets directly within Docker containers. This facilitates the verification of protocol-compliant message exchanges, the calculation of actual network traffic generated by each configuration, and the validation of handshake duration measurements based on packet transmission timestamps.

Orchestration and Analysis Layer: this layer is responsible for managing scenario execution and coordinating all components of the framework via a set of `bash` scripts. Execution logs and captured network traffic are processed using `Python` scripts, which generate CSV files containing the relevant data. These files serve as the basis for generating plots and performing statistical analyses for each evaluated scenario. In addition to basic statistical metrics, our analysis includes the computation of the coefficient of variation, and outliers.

The evaluation framework is available at `https://github.com/montenegro-montes/TLS-VPN`. The repository contains all the necessary files to run the platform, including Docker installation instructions, scenario execution scripts, and analysis tools. Additionally, it provides access to the logs and plots generated and used in this work.

3.3 Hardware Platforms and Execution Environment

The evaluation was conducted on two hardware platforms: an ARM-based system and an x86-based system. The ARM platform is a Macbook Pro 13 fitted with an Apple M2 System on Chip that offers 8 cores: 4 performance cores (up to 3.5 GHz), and 4 power-efficiency cores (up to 2,4 GHz). The x86 system is a server with two Intel Xeon Silver 4214 CPUs, each with 12 physical cores running at 2.20 GHz, but it can run at Turbo speed (3.20 GHz) on demand. The experiments were run on a virtualized Ubuntu Desktop machine with 24 logical cores running on top of the VMware ESXi version 8.0 hypervisor. The x86 processor supports advanced vector instruction sets, including AVX, AVX2, and AVX-512 (F, DQ, BW, and VL extensions). No CPU frequency scaling or turbo boost mechanisms were active during the experiments, ensuring stable and deterministic performance measurements.

Both platforms running `Ubuntu 24.10`, aligned with cloud-native and container-oriented deployment practices. To ensure reproducibility and prevent ABI incompatibilities among cryptographic libraries, the complete evaluation environment is encapsulated within a `Docker` container. The cryptographic stack relies on `OpenSSL 3.4` and `liboqs v0.12.0`, with `oqsprovider v0.8.0` acting as the integration layer between both components. All software is compiled using `GCC 14.2.0` with using `-O3` optimization.

A core design principle of the framework was to minimize modifications to standard software distributions. This approach ensures compatibility with upstream updates and avoids introducing dependencies on custom or patched software components, thereby enhancing maintainability and ease of integration. Although only standard software distributions are used, all components are locally compiled, enabling platform-specific optimizations based on the underlying hardware and available CPU instruction sets.

For the evaluation of cryptographic primitives, each benchmark consisted of fifty consecutive executions of the `openssl speed` command, automated through the framework. All signature and KEM operations were executed within a containerized environment configured with strict CPU and I/O isolation to minimize system-induced variability. For each primitive, fifty latency samples were collected to compute the mean execution time.

For the measurement of TLS handshake duration, we developed a minimal custom program based on `s_client` from the OpenSSL toolkit. This tool establishes multiple handshake connections within a configurable time window and reports the mean duration of successful handshakes over that period. Although this approach allows for automated and repeatable measurements, it introduces a relatively high degree of variability due to concurrent execution and internal timing mechanisms.

For the application-level measurements, we adhered to the principle of avoiding any modifications to the original OpenVPN distribution. Consequently, the evaluation was conducted by capturing network traffic and analyzing packet timestamps to determine connection establishment durations. This strategy enhances the framework's adaptability to future versions of the application, as it avoids introducing software dependencies or requiring changes to the source code. However, it also incurs additional delays due to the overhead introduced by packet capture and logging mechanisms. This effect becomes particularly evident when comparing the TLS handshake durations measured during VPN connection setup with those obtained using the custom measurement tool.

3.4 Metrics and Statistical Analysis

In all experiments, in addition to computing the mean and standard deviation (SD) of the connection duration, we also calculate the median and Interquartile Range (IQR) as well as the coefficient of variation (CV), and the percentage of outliers. We use Tukey's method for the identification of outliers, which establish as outliers those values outside $1.5 \times$ IQR below the first quartile or above the third quartile. All our results are obtained from continuous executions ('hot' runs), as suggested by [25], and warm-up runs are excluded from the reported data.

The mean and median are measures of central tendency and comparing them indicate the skewness (asymmetry) of the distribution. The SD and IQR quantify spread in the original data units, while the CV normalizes this spread. In case the SD is disproportionately large compared to the IQR, it confirms that the outliers are heavily influencing the overall variability. The CV, calculated as the ratio of the SD to the mean, is a unitless measure often express as a percentage.

A low CV (below 10%) indicates stable and predictable handshake timing, whereas higher values reveal greater variability and jitter, which could impact the reliability of real-world handshake performance.

4 Cryptographic Primitives Evaluation

This section evaluates the execution time of key encapsulation/exchange mechanisms (KEMs) and signature schemes across two processor architectures—ARM and

x86— quantifying the computational overhead of each primitive.

The evaluation consists of fifty consecutive runs of OpenSSL's `speed` command for each of the algorithms introduced in Sec. 3.1. These executions were launched using the evaluation framework inside a container with strict CPU and I/O isolation. Precise evaluation details are provided in Sec. 3.3.

4.1 Benchmarking KEM primitives

Key encapsulation mechanisms consist of three main operations: (i) `KeyGen`, a probabilistic algorithm that generates a public/private key pair (`pk`, `sk`); (ii) `Encaps`, a probabilistic algorithm that takes `pk` as input and outputs a ciphertext `ct` along with a shared secret `ss`; and (iii) `Decaps`, a deterministic algorithm that recovers `ss` from `sk` and `ct`. The results of executing these operations on both ARM and x86 architectures are detailed in Table 2. Additional statistical metrics, including median, IQR and outlier percentage, are provided in Table A.1 in Appendix A.

The results show that ARM outperforms x86 across all cryptographic algorithms and security levels, except for `ML-KEM`, which consistently runs faster on x86—by a factor of 1.5 to 2.5× depending on the primitive. In contrast, execution times exhibited by `HQC`, the other post-quantum alternative, are up to 3 times better on ARM but they are more stable on x86, as indicated by dispersion metrics, particularly the coefficient of variation (CV), which is unitless and therefore well suited for comparing relative variability across distributions with different scales. In general, the CV stays below 1% on x86—except for `ML-KEM` which stays below 4%—while, on ARM, it typically falls within 4.85% to 5.15% (95% confidence interval). Outliers are generally limited —typically below 10% (see Table A.1)— although higher outlier percentages appear more frequently on ARM. Notably, `ML-KEM` exhibit the highest percentage of outliers; however, this is largely attributable to its extremely low central dispersion (IQR), particularly in x86, causing minor timing deviations to be classified as outliers despite highly stable central behavior.

Among all primitives, `HQC` variants exhibit the most stable behavior, yet also the highest execution times, with an overhead of up to 100× on ARM—compared to `X25519` Decaps—and even higher on x86. The computational cost of `HQC` is so significant relative to other algorithms that its combination with elliptic curves (i.e., hybrid `HQC`) has a negligible impact on overall execution time. This is not the case for hybrid `ML-KEM`, whose execution time increases by up to a factor of 25 when combined with `P521`.Nonetheless, the execution performance of hybrid `ML-KEM` is significantly superior to that of plain `HQC`.

The transition from Level I to Level III reveals different behaviors. Classical ECDH-based schemes experience the steepest increase on both ARM and x86 architectures: KeyGen slows down by approximately 6×, Encaps by 5× and Decaps by 4×. `ML-KEM` shows a modest latency increase of approximately 1.6× across all operations, whereas `HQC` exhibits a more significant increase of 2.5–3×. Hybrid `ML-KEM` combines the costs of both components, resulting in a 4–5× penalty, while hybrid `HQC` closely mirrors the increase of pure `HQC`.

Moving from Level III to Level V exacerbates these trends. Classical schemes slow down significantly—KeyGen and Encaps approximately 10×, and Decaps up to 12×. In contrast, both `ML-KEM` and the pure and hybrid variants of `HQC` remain relatively stable, with execution times increasing by only 1.4–2×. However, hybrid `ML-KEM` incurs a 9–11× increase.

Overall, these results highlight the favorable overhead and scalability characteristics of `ML-KEM` compared to classical and other post-quantum KEMs. However, hybrid deployments demand careful evaluation due to their compounded overheads. Another important aspect is the improved efficiency of `ML-KEM` on x86 architectures, where it performs approximately 2× faster than on ARM.

4.2 Benchmarking signature primitives

Similar to KEMs, signature algorithms involve three main operations: (i) `KeyGen`, a probabilistic algorithm that generates a secret key `sk` for signing and a public key for verification `pk`; (ii) `Sign`, an algorithm that receives a message `m` and a secret key `sk` and outputs a signature σ; and (iii) `Verify`, an algorithm that takes a message `m`, a public key `pk` and a signature σ, and outputs a bit b indicating whether the signature is valid. Table 3 presents the execution times of the primitives on both ARM and x86 architectures. Additional statistical metrics are provided in Table A.2 in Appendix A.

The results indicate that, for elliptic curves, ARM consistently outperforms x86 by roughly a factor of 2-3× for all cryptographic operations. The performance of signatures and verification is stable and predictable on both architectures as indicated by a relatively low CV. However, the execution times and CV of the key generation operation (`KeyGen`) are significantly higher and less stable, but still exhibits a low IQR and outliers percentage (as shown in Table A.2). This is generally not a concern, as this primitive is only barely used in practice.

Table 2: Comparison of KEM operations times on ARM and x86 architectures. SpeedUp values above 1 indicate a performance improvement, whereas values below 1 indicate a slowdown.

Security Level	Signature Primitive	ARM			x86			SpeedUp ARM vs x86		
		KeyGen	Encaps	Decaps	KeyGen	Encaps	Decaps			
		Mean±SD/CV	Mean±SD/CV	Mean±SD/CV	Mean±SD/CV	Mean±SD/CV	Mean±SD/CV	KeyG	Enc	Dec
I	x25519	0.024 ± 0.001 2.17	0.055 ± 0.002 3.41	0.030 ± 0.001 4.81	0.047 ± 0.001 2.07	0.106 ± 0.001 0.96	0.053 ± 0.000 0.69	2.00×	1.93×	1.77×
III	x448	0.153 ± 0.005 3.51	0.267 ± 0.009 3.45	0.112 ± 0.004 3.98	0.304 ± 0.003 0.87	0.553 ± 0.005 0.95	0.243 ± 0.002 0.94	1.99×	2.07×	2.17×
V	P-521	1.333 ± 0.048 3.63	2.675 ± 0.079 2.94	1.361 ± 0.091 6.72	2.820 ± 0.021 0.75	5.676 ± 0.049 0.86	2.842 ± 0.030 1.04	2.12×	2.12×	2.09×
I	x25519_mlkem512	0.045 ± 0.002 3.82	0.076 ± 0.002 3.20	0.081 ± 0.003 3.44	0.070 ± 0.001 1.28	0.124 ± 0.002 1.29	0.122 ± 0.002 1.51	1.58×	1.64×	1.50×
I	mlkem512	0.020 ± 0.001 2.66	0.022 ± 0.000 2.11	0.027 ± 0.000 1.82	0.013 ± 0.001 3.74	0.012 ± 0.000 1.99	0.011 ± 0.000 3.84	0.67×	0.55×	0.41×
III	x448_mlkem768	0.200 ± 0.007 3.40	0.322 ± 0.010 3.17	0.326 ± 0.012 3.54	0.368 ± 0.003 0.91	0.632 ± 0.006 0.93	0.632 ± 0.005 0.85	1.84×	1.96×	1.94×
III	mlkem768	0.034 ± 0.002 4.47	0.036 ± 0.001 3.83	0.044 ± 0.002 5.11	0.021 ± 0.000 1.97	0.019 ± 0.000 0.74	0.018 ± 0.000 2.67	0.62×	0.53×	0.42×
V	p521_mlkem1024	1.899 ± 0.049 2.60	2.789 ± 0.118 4.23	1.596 ± 0.067 4.20	3.971 ± 0.034 0.86	5.816 ± 0.069 1.18	3.304 ± 0.029 0.88	2.09×	2.09×	2.07×
V	mlkem1024	0.052 ± 0.002 4.69	0.052 ± 0.002 4.32	0.063 ± 0.004 5.99	0.029 ± 0.001 1.84	0.027 ± 0.000 1.54	0.027 ± 0.000 1.66	0.57×	0.52×	0.43×
I	x25519_hqc128	0.901 ± 0.031 3.44	1.845 ± 0.072 3.92	3.081 ± 0.098 3.18	2.619 ± 0.023 0.88	5.301 ± 0.050 0.94	8.367 ± 0.086 1.03	2.91×	2.87×	2.72×
I	hqc128	0.906 ± 0.036 3.93	1.818 ± 0.066 3.61	3.029 ± 0.069 2.29	2.560 ± 0.022 0.87	5.179 ± 0.049 0.95	8.260 ± 0.074 0.89	2.83×	2.85×	2.73×
III	x448_hqc192	2.792 ± 0.099 3.56	5.562 ± 0.183 3.29	8.688 ± 0.289 3.33	8.141 ± 0.068 0.83	16.293 ± 0.140 0.86	24.807 ± 0.195 0.79	2.92×	2.93×	2.86×
III	hqc192	2.776 ± 0.057 2.07	5.459 ± 0.126 2.30	8.722 ± 0.226 2.59	7.839 ± 0.070 0.90	15.713 ± 0.130 0.82	24.257 ± 0.245 1.01	2.82×	2.88×	2.78×
V	p521_hqc256	6.507 ± 0.098 1.51	12.125 ± 0.093 0.77	16.866 ± 0.132 0.78	18.101 ± 0.156 0.86	34.186 ± 0.264 0.77	47.522 ± 0.339 0.71	2.78×	2.82×	2.82×
V	hqc256	4.708 ± 0.061 1.29	9.506 ± 0.069 0.72	15.410 ± 0.048 0.31	14.325 ± 0.139 0.97	28.819 ± 0.277 0.96	44.722 ± 0.374 0.84	3.04×	3.03×	2.90×

Table 3: Comparison of Signature operations times on ARM and x86 architectures.

Security Level	Signature Primitive	ARM			x86			SpeedUp ARM vs x86		
		KeyGen	Sign	Verify	KeyGen	Sign	Verify			
		Mean±SD/CV	Mean±SD/CV	Mean±SD/CV	Mean±SD/CV	Mean±SD/CV	Mean±SD/CV	KeyG	Sign	Verif
I	ed25519	4.191 ± 0.137 3.260	0.024 ± 0.000 1.550	0.077 ± 0.002 2.180	10.930 ± 0.932 8.530	0.048 ± 0.001 1.130	0.156 ± 0.002 1.120	2.61×	1.98×	2.03×
III	ecdsap384	5.360 ± 0.140 5.378	0.562 ± 0.007 1.240	0.470 ± 0.006 1.270	13.026 ± 0.862 6.620	1.253 ± 0.012 0.990	1.034 ± 0.010 1.000	2.43×	2.23×	2.20×
V	ecdsap521	7.031 ± 0.152 2.160	1.405 ± 0.018 1.320	1.103 ± 0.037 3.330	15.846 ± 0.446 2.820	3.001 ± 0.046 1.520	2.351 ± 0.036 1.510	2.25×	2.14×	2.13×
I	mldsa44	4.220 ± 0.587 13.920	0.235 ± 0.002 0.950	0.060 ± 0.002 3.360	10.742 ± 0.687 6.400	0.110 ± 0.001 0.850	0.039 ± 0.000 0.610	2.54×	0.46×	0.65×
III	mldsa65	4.517 ± 0.173 3.830	0.383 ± 0.005 1.340	0.096 ± 0.004 4.150	10.805 ± 0.733 6.780	0.176 ± 0.001 0.500	0.066 ± 0.000 0.760	2.39×	0.46×	0.68×
V	mldsa87	4.503 ± 0.163 3.630	0.460 ± 0.004 0.920	0.157 ± 0.005 3.180	10.905 ± 0.479 4.390	0.221 ± 0.001 0.600	0.105 ± 0.001 0.510	2.42×	0.48×	0.67×

In contrast, post-quantum ML-DSA exhibits a different performance profile. While ARM remains faster in key generation, it is slower than x86 (by a factor of 2-3×) in signing and verification across all security levels. This is consistent with the results of ML-KEM in Sec. 4.1, which may suggest that post-quantum algorithms based on modular lattices can more effectively leverage AVX2-enabled SIMD parallelism on x86 processors. Works such as [17] report performance improvements enabled by AVX2 and AVX-512 vector instructions over non-vectorized code. While AVX-512 provides additional benefits over AVX2, the major performance gains stem from the adoption of AVX2, with reported speedups between 6× and 9×, depending on the security level and the ML-KEM operation.

Regarding scalability, moving across security levels reveals different latency patterns between traditional elliptic curve-based signatures and post-quantum signature schemes. Elliptic curves exhibit a two-stage growth pattern: the most significant increase occurs when moving from Level I to Level III on both ARM and x86, where signing time increases by more than one order of magnitude and verification nearly reaches one order of magnitude. The subsequent transition to Level V

leads to a further 2–3× increase in both operations. In contrast, `ML-DSA` shows a more gradual increase, with signing times roughly doubling and verification times tripling from Level I to Level V, on both architectures. Key generation times remain largely unaffected across security levels.

5 Performance Evaluation in TLS Handshake

This section evaluates post-quantum cryptographic primitives when integrated into the Transport Layer Security (TLS) protocol. Prior to analyzing mutually authenticated TLS handshake execution times, we present an overview of the TLS handshake, describe the cryptographic primitives involved, and provide a preliminary assessment of their cost within the protocol.

5.1 Overview of TLS Handshake

The TLS protocol is a fundamental Internet security protocol. It consists of a handshake phase followed by a data transmission phase. During the handshake phase, public-key cryptography is used for key exchange and authentication. The resulting key material is then used in the data transmission phase to secure communication using symmetric-key cryptography [26]. Consequently, our focus is on the handshake phase and its associated cryptographic operations.

Fig. 2 illustrates the message flow and cryptographic operations in a mutually authenticated TLS handshake, based on certificates signed by a certificate authority. This mode requires at least nine cryptographic operations, with additional certificate verifications potentially required depending on the depth of the certificate chain. We focus on mutually authenticated TLS as we later investigate the performance of TLS-based VPNs, which also rely on mutual authentication. An analysis of TLS with server-only authentication is presented in [14].

During the key exchange sub-phase, the client performs `KeyGen` and `Decaps`, while the server executes `Encaps`. During the authentication subphase, both parties sign the handshake transcript and each performs three verification operations—one for the peer's handshake signature and two for validating the certificate chain. We used `ltrace` to validate that each party executed the expected primitives according to the protocol specification, and to measure the execution time of the cryptographic primitives invoked during the handshake.

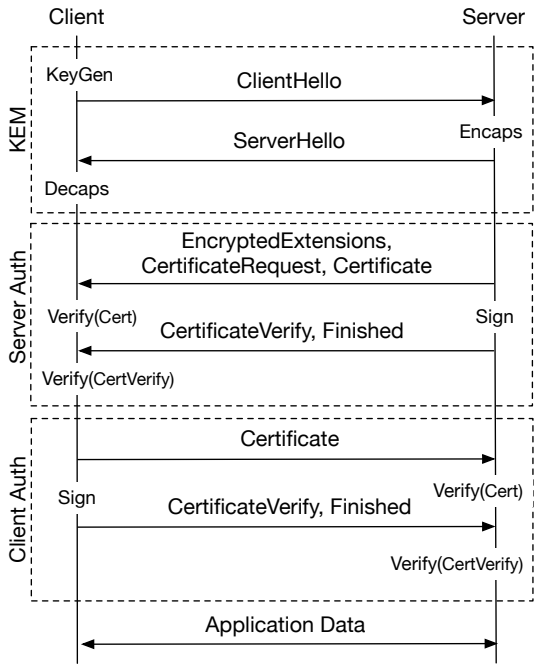


Fig. 2: Cryptographic operations involved in a mutually authenticated TLS handshake

5.2 Mutually Authenticated TLS Handshake Evaluation

This section analyzes the execution times of mutually authenticated TLS handshakes —based on OpenSSL v3.4— under ideal network conditions. While the evaluation framework supports realistic network conditions, an ideal setting was used to isolate the impact of platform architecture differences and to analyze the time distribution between cryptographic and non-cryptographic operations.

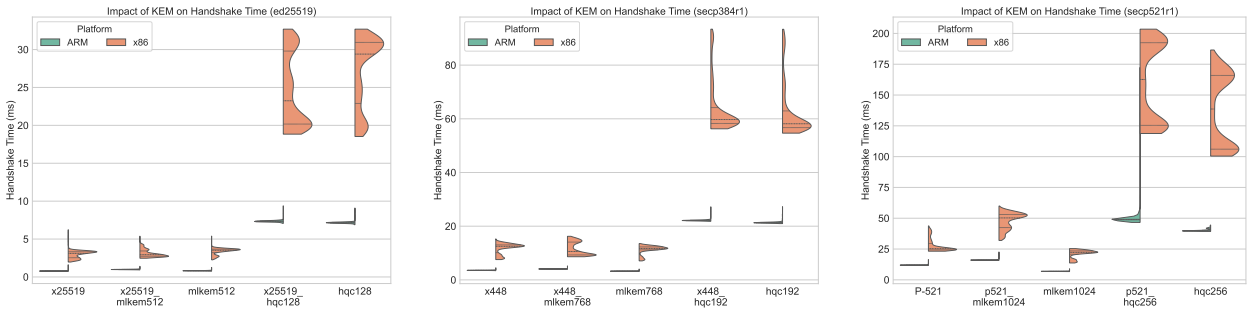
Regarding the evaluation of TLS configurations involving *traditional signatures* (Table 4 and Fig. 3)[1], the results show a consistent performance advantage for ARM across all KEMs and security levels, with *SpeedUp* values ranging from 2.32 to 4.07. These results reinforce the trend observed in Sec. 5.3. In general, traditional primitives and those involving `ML-KEM` exhibit similar and stable handshake durations across all security levels. In contrast, configurations using `HQC` result in a significant handshake time increase and variability. This cost becomes prohibitive on both platforms—particularly on x86—when using Level V `HQC` variants.

The results of running TLS handshakes with *post-quantum signatures* are presented in Table 5 and Fig. 4.

[1] Figures are included to illustrate the distribution of execution times and highlight performance differences and variability across platforms. The signature algorithm used at each security level is indicated at the top of the subfigures.

Table 4: Handshake times (ms) for mutually authenticated TLS handshakes with traditional signatures. "SpeedUp" is the performance advantage of ARM over x86.

Security Level	KEM	ARM					x86					SpeedUp
		Mean ± SD	Median	IQR	CV	Out. %	Mean ± SD	Median	IQR	CV	Out. %	
I	x25519	0.78 ± 0.04	0.78	0.02	0.05	3.30	2.96 ± 0.47	3.11	0.79	0.16	0.20	3.79×
	x25519_mlkem512	0.99 ± 0.03	0.99	0.03	0.03	2.05	3.10 ± 0.48	2.94	0.70	0.15	0.10	3.14×
	mlkem512	0.83 ± 0.03	0.82	0.03	0.03	1.45	3.36 ± 0.42	3.52	0.45	0.12	6.05	4.07×
	x25519_hqc128	7.34 ± 0.10	7.34	0.11	0.01	1.50	24.99 ± 4.95	23.80	10.04	0.20	0.00	3.40×
	hqc128	7.16 ± 0.10	7.16	0.10	0.01	1.70	27.40 ± 4.72	29.62	7.94	0.17	0.00	3.83×
III	x448	3.55 ± 0.05	3.55	0.04	0.01	4.45	11.73 ± 1.95	12.50	2.89	0.17	0.00	3.30×
	x448_mlkem768	4.08 ± 0.06	4.07	0.05	0.01	2.90	11.55 ± 2.41	10.58	4.77	0.21	0.00	2.83×
	mlkem768	3.24 ± 0.05	3.24	0.04	0.01	5.65	11.28 ± 1.56	11.71	1.09	0.14	15.90	3.48×
	x448_hqc192	22.16 ± 0.23	22.13	0.17	0.01	3.95	66.26 ± 12.45	60.04	11.29	0.19	13.68	2.99×
	hqc192	21.40 ± 0.46	21.31	0.18	0.02	6.00	63.63 ± 11.22	58.29	7.93	0.18	14.14	2.97×
V	P-521	11.96 ± 0.21	11.93	0.12	0.02	4.30	27.76 ± 5.15	25.32	5.12	0.19	10.61	2.32×
	p521_mlkem1024	16.05 ± 0.28	16.01	0.13	0.02	5.55	47.78 ± 6.62	50.26	10.59	0.14	0.00	2.98×
	mlkem1024	6.89 ± 0.10	6.88	0.10	0.01	2.00	21.16 ± 3.31	22.28	2.72	0.16	19.36	3.07×
	p521_hqc256	49.78 ± 6.53	48.92	0.45	0.13	12.41	165.59 ± 33.69	182.00	70.10	0.20	0.00	3.33×
	hqc256	40.24 ± 0.99	39.82	0.61	0.02	17.45	137.57 ± 29.37	138.65	59.85	0.21	0.00	3.42×



(a) Level I (b) Level III (c) Level V

Fig. 3: TLS handshake using traditional signatures on ARM and x86

These settings consider future scenarios where both the KEM and signature primitives are post-quantum. Traditional KEMs are included solely as a baseline reference. Although in these settings TLS handshakes continue to perform better in ARM, the *SpeedUp* values are smaller—ranging from 1.33 to 2.60. This behavior can be partly explained because post-quantum `ML-DSA` signature performs better on x86, as shown in Table 3.

Finally, the influence of post-quantum signature algorithms lead to narrower confidence intervals and CV values on x86 compared to ARM, reversing the trend observed in a setting with traditional signatures. This shift aligns with the prior evaluation of signature primitives, where x86 demonstrated better performance than ARM for certain post-quantum schemes.

5.3 Execution time of TLS handshake primitives

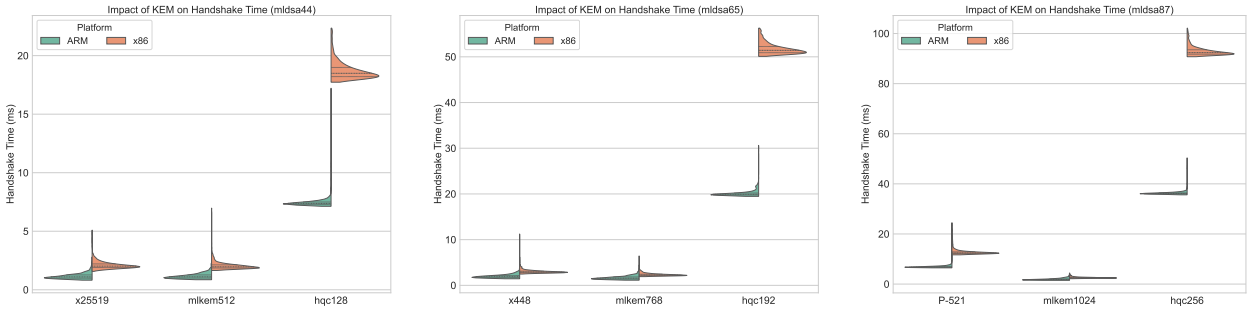
This section examines the proportion of the overall TLS handshake execution time attributable to cryptographic primitives.

A baseline approach to tackle this task while minimizing code changes is to make an estimate based on the aggregated execution times of the individual primitives reported in Tables 2 and 3. This cost is represented under the "baseline" columns in Tables 6 and 7, which compare the overall TLS handshake time and the aggregated cryptographic execution time, reporting their ratio.

These results reveal a significant portion of the handshake on ARM is due to cryptographic operations. Despite the lower relative cryptographic cost on x86, the total handshake time remains significantly higher across all configurations. Moreover, cryptographic operations exhibit lower variability on ARM.

Table 5: Handshake times (ms) for mutually authenticated TLS handshakes with post-quantum signatures. "SpeedUp" is the performance advantage of ARM over x86.

Security Level	KEM	ARM					x86					SpeedUp
		Mean ± SD	Median	IQR	CV	Out. %	Mean ± SD	Median	IQR	CV	Out. %	
I	x25519	1.16 ± 0.24	1.10	0.28	0.20	3.35	2.09 ± 0.31	2.02	0.31	0.15	5.25	1.81×
	mlkem512	1.16 ± 0.25	1.10	0.27	0.22	2.80	2.04 ± 0.30	1.96	0.27	0.15	6.25	1.76×
	hqc128	7.43 ± 0.32	7.38	0.21	0.04	4.40	19.12 ± 1.83	18.55	0.94	0.10	9.76	2.57×
III	x448	1.97 ± 0.42	1.90	0.45	0.21	3.60	3.01 ± 0.37	2.92	0.33	0.12	6.35	1.52×
	mlkem768	1.67 ± 0.40	1.57	0.47	0.24	3.15	2.34 ± 0.31	2.26	0.30	0.13	6.50	1.40×
	hqc192	20.13 ± 0.75	19.93	0.43	0.04	9.85	52.42 ± 3.16	51.51	1.63	0.06	8.56	2.60×
V	P-521	6.90 ± 0.48	6.80	0.37	0.07	4.25	12.79 ± 1.20	12.43	0.65	0.09	9.25	1.86×
	mlkem1024	1.93 ± 0.37	1.82	0.43	0.19	3.90	2.55 ± 0.28	2.48	0.26	0.11	5.75	1.33×
	hqc256	36.29 ± 0.70	36.16	0.42	0.02	4.30	94.28 ± 5.91	92.51	2.12	0.06	10.89	2.60×



(a) Level I (b) Level III (c) Level V

Fig. 4: TLS handshake using post-quantum signatures on ARM and x86.

These results suggest that non-cryptographic operations may introduce additional overhead on x86 under ideal conditions, and that x86 generally shows greater performance variability.

To validate these calculations without altering the codebase, we rely on `ltrace`[2], which allows us to monitor the invoked cryptographic functions and measure their execution time. This enables an independent estimation of the computational cost of each primitive. For the percentage calculation, the aggregated primitive cost is compared against the handshake execution time obtained through `ltrace`, since the use of this tool introduces a significant overhead, as shown in the "ltrace" column in Tables 6 and 7.

The baseline evaluations and the measurements obtained with `ltrace` exhibit strong correspondence on ARM for both classical and post-quantum signatures, whereas on x86 the discrepancies between the two become more pronounced. This behavior is primarily due to the optimizations that x86 applies when executing cryptographic primitives in isolation, as well as the additional overhead introduced by non-cryptographic tasks during real handshake execution.

Focusing on the comparison between platform architectures and `ltrace` measurement, ARM spends between 39% and 61% of the handshake time on cryptographic operations, corresponding to Security Levels I and V with traditional signatures, respectively. In contrast, x86 allocates between 41% and 64%. Regarding handshakes involving post-quantum signatures, both ARM and x86 exhibit very similar behavior across security levels, with ratios ranging approximately from 40% up to 57%.

These results indicate that, despite architectural differences, the relative cryptographic cost of post-quantum signatures is remarkably consistent across ARM and x86. Traditional signatures, however, show greater variability between platforms, suggesting that platform specific optimizations in x86 amplify non-cryptographic overheads. Overall, post-quantum signatures introduce a stable and predictable computational footprint that is less sensitive to architectural differences.

[2] ltrace: library call tracer - Linux man page. `https://www.ltrace.org/`

Table 6: TLS connection establishment with traditional signatures and the evaluation of handshake time spent on cryptographic operations

Security Level	KEM	ARM						x86					
		baseline			ltrace			baseline			ltrace		
		TLS ± SD	Crypto ± SD	Crypto %	TLS ± SD	Crypto ± SD	Crypto %	TLS ± SD	Crypto ± SD	Crypto %	TLS ± SD	Crypto ± SD	Crypto %
I	x25519	0.78 ± 0.04	0.31 ± 0.01	38.79	5.65 ± 0.19	2.24 ± 0.12	39.63	2.96 ± 0.47	0.62 ± 0.01	21.26	6.50 ± 0.64	2.67 ± 0.30	41.05
	x25519_mlkem512	0.99 ± 0.03	0.38 ± 0.01	37.83	7.40 ± 0.38	4.00 ± 0.34	54.00	3.10 ± 0.48	0.71 ± 0.01	21.27	8.37 ± 0.50	4.45 ± 0.30	53.17
	mlkem512	0.83 ± 0.03	0.30 ± 0.01	35.85	6.17 ± 0.24	2.86 ± 0.16	46.36	3.36 ± 0.42	0.54 ± 0.01	17.33	7.30 ± 0.57	3.44 ± 0.28	47.10
	x25519_hqc128	7.34 ± 0.10	4.24 ± 0.10	58.28	13.94 ± 0.58	8.00 ± 0.31	57.42	24.99 ± 4.95	11.50 ± 0.09	39.32	26.22 ± 2.47	15.04 ± 1.77	57.30
	hqc128	7.16 ± 0.10	4.19 ± 0.09	59.34	12.70 ± 0.64	6.85 ± 0.41	53.93	27.40 ± 4.72	11.34 ± 0.08	41.73	25.55 ± 2.88	14.13 ± 1.67	55.31
III	x448	3.55 ± 0.05	2.24 ± 0.02	63.43	8.87 ± 0.20	4.64 ± 0.16	52.28	11.73 ± 1.95	4.90 ± 0.03	48.85	13.18 ± 2.33	7.31 ± 1.33	55.43
	x448_mlkem768	4.08 ± 0.06	2.50 ± 0.02	61.44	10.93 ± 0.22	6.51 ± 0.19	59.51	11.55 ± 2.41	5.36 ± 0.03	41.77	16.23 ± 1.55	9.66 ± 1.10	59.44
	mlkem768	3.24 ± 0.05	2.05 ± 0.02	63.56	9.08 ± 0.22	5.10 ± 0.16	56.19	11.28 ± 1.56	4.39 ± 0.03	43.88	12.84 ± 1.37	7.48 ± 0.96	58.19
	x448_hqc192	22.16 ± 0.23	13.45 ± 0.37	60.60	29.04 ± 0.24	17.51 ± 0.20	60.28	66.26 ± 12.45	37.30 ± 0.20	45.77	67.67 ± 5.30	40.14 ± 3.31	59.34
	hqc192	21.40 ± 0.46	13.47 ± 0.23	62.59	27.07 ± 0.27	15.95 ± 0.20	58.94	63.63 ± 11.22	36.45 ± 0.26	50.04	63.22 ± 3.39	37.52 ± 2.65	59.32
V	P-521	11.96 ± 0.21	7.41 ± 0.15	61.69	17.58 ± 0.24	10.15 ± 0.17	57.75	27.76 ± 5.15	15.72 ± 0.14	49.42	30.06 ± 1.74	17.68 ± 1.15	58.83
	p521_mlkem1024	16.05 ± 0.28	8.21 ± 0.14	51.07	23.68 ± 0.82	14.23 ± 0.60	60.06	47.78 ± 6.62	17.33 ± 0.15	40.16	41.33 ± 2.58	24.91 ± 1.75	60.28
	mlkem1024	6.89 ± 0.10	4.83 ± 0.11	69.90	12.79 ± 0.25	7.86 ± 0.20	61.44	21.16 ± 3.31	10.11 ± 0.14	48.42	20.18 ± 2.17	12.93 ± 1.46	64.08
	p521_hqc256	49.78 ± 6.53	28.09 ± 0.20	56.99	56.38 ± 0.55	34.47 ± 0.36	61.13	165.59 ± 33.69	75.68 ± 0.46	48.12	132.95 ± 8.15	79.67 ± 4.74	59.95
	hqc256	40.24 ± 0.99	24.83 ± 0.14	61.55	45.68 ± 0.56	28.09 ± 0.37	61.50	137.57 ± 29.37	69.10 ± 0.42	53.70	112.98 ± 7.32	68.59 ± 5.60	60.69

Table 7: TLS connection establishment with post-quantum signatures and the evaluation of handshake time spent on cryptographic operations

Security Level	KEM	ARM						x86					
		baseline			ltrace			baseline			ltrace		
		TLS ± SD	Crypto ± SD	Crypto %	TLS ± SD	Crypto ± SD	Crypto %	TLS ± SD	Crypto ± SD	Crypto %	TLS ± SD	Crypto ± SD	Crypto %
I	x25519	1.16 ± 0.24	0.47 ± 0.01	40.41	6.02 ± 0.21	2.49 ± 0.18	41.30	2.09 ± 0.31	0.33 ± 0.00	16.08	7.21 ± 1.09	2.91 ± 0.46	40.37
	mlkem512	1.16 ± 0.25	0.46 ± 0.01	42.80	6.50 ± 0.28	3.00 ± 0.23	46.18	2.04 ± 0.30	0.25 ± 0.00	11.83	7.79 ± 0.85	3.53 ± 0.48	45.28
	hqc128	7.43 ± 0.32	4.35 ± 0.09	58.51	12.96 ± 0.31	6.93 ± 0.25	53.45	19.12 ± 1.83	11.05 ± 0.08	57.37	25.83 ± 3.23	14.20 ± 2.08	54.88
III	x448	1.97 ± 0.42	0.94 ± 0.02	50.76	6.90 ± 0.38	3.10 ± 0.30	44.76	3.01 ± 0.37	0.92 ± 0.00	29.78	8.31 ± 1.17	3.57 ± 0.59	42.89
	mlkem768	1.67 ± 0.40	0.75 ± 0.01	47.98	6.94 ± 0.37	3.37 ± 0.30	48.46	2.34 ± 0.31	0.41 ± 0.00	17.63	7.84 ± 1.01	3.58 ± 0.48	45.68
	hqc192	20.13 ± 0.75	12.17 ± 0.23	61.22	26.30 ± 1.17	14.60 ± 0.69	55.51	52.42 ± 3.16	32.47 ± 0.25	61.75	61.59 ± 4.43	34.63 ± 3.20	56.21
V	P-521	6.90 ± 0.48	3.63 ± 0.10	52.00	12.45 ± 0.46	6.05 ± 0.38	48.57	12.79 ± 1.20	6.20 ± 0.04	48.65	18.71 ± 1.80	8.74 ± 0.79	46.81
	mlkem1024	1.93 ± 0.37	1.05 ± 0.02	55.81	7.59 ± 0.32	3.69 ± 0.31	48.62	2.55 ± 0.28	0.59 ± 0.00	23.11	8.63 ± 0.85	3.93 ± 0.43	45.53
	hqc256	36.29 ± 0.70	21.05 ± 0.08	58.15	42.22 ± 0.46	24.15 ± 0.39	57.20	94.28 ± 5.91	59.58 ± 0.39	63.06	105.58 ± 8.26	60.58 ± 6.30	57.31

6 Performance Evaluation in TLS-based VPNs

After evaluating the cryptographic primitives on both platforms and analyzing their impact within the TLS handshake, we now extend the study to a real-world application that relies on TLS. Specifically, we focus on VPNs, selecting OpenVPN as the target due to its wide adoption and native support for TLS.

Firstly, a technical overview of the OpenVPN protocol is introduced, providing the necessary context to understand system deployment and evaluation. Subsequently, the evaluation is conducted under two scenarios: one representing the current state with traditional signature schemes, and another envisioning a quantum future that employs post-quantum signatures.

6.1 OpenVPN Protocol Overview

OpenVPN is a widespread, open-source protocol for establishing tunnels over TCP or UDP [27]. OpenVPN encrypted tunnels can be based on a pre-shared static key or public key certificates, using the Transport Layer Security (TLS) protocol for session establishment, which is the default option.

The message structure defined by OpenVPN consists of a variable-length packet header and a payload. The packet header indicates the message type along with some optional fields to control de information flow, especially when OpenVPN is transported over UDP. Two main categories of messages are defined: control and data messages. Control messages (`P_CONTROL_*` and `P_ACK`) are used for tunnel management, authentication, and key exchange, while data messages (`P_DATA`) carry the actual VPN traffic payload.

The protocol operation is illustrated in Fig. 5. First, the client and server agree on the key derivation method to be used. Subsequently, `P_CONTROL_V1` messages encapsulate the TLS handshake between the client and server. This handshake provides entity authentication and establishes a shared master secret. The TLS master secret is then used to derive the keys—according to the previously agreed method—to secure data transmission through the tunnel. By default, modern versions of OpenVPN derive four symmetric keys from the master secret: two for encryption and two for integrity, with one pair allocated for each communication direction,

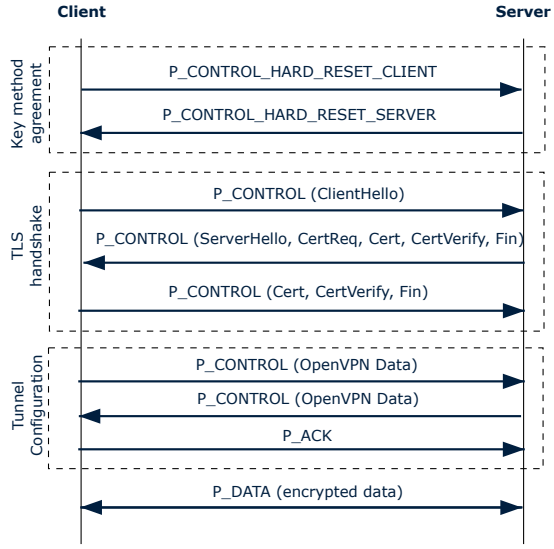


Fig. 5: OpenVPN Protocol

preventing key reuse. Older versions used only one pair of keys.

Next, the client and the server exchange parameters for the configuration tunnel using (e.g., cryptographic primitives to be used) using `P_CONTROL` messages. Finally, application data can be securely exchanged encrypted using the keys derived within `P_DATA` messages.

6.2 OpenVPN Handshake Evaluation

The evaluation of OpenVPN focused on the establishment phase as this is where post-quantum cryptographic primitives are used. We used OpenVPN v2.6.12 and relied on traffic capture tools—specifically `Wireshark` and `Edgeshark`—provided by the network layer of our evaluation framework to calculate the VPN establishment time and the duration of the embedded TLS handshake.

The results of our evaluation are presented in Tables 8 and 9 (and Fig. 6 and 7), which correspond to current and future fully post-quantum scenarios, respectively. These tables report some key statistical metrics relative to the overall VPN execution time—mean, standard deviation, median, IQR, CV and outliers—along with the mean duration of the TLS handshake during the VPN connection phase on both platforms. Additionally, the "SpeedUp" columns quantify the execution time improvement of ARM over x86 for both the overall VPN establishment and the embedded TLS handshake.

In the current situation with *traditional signatures* (Table 8 and Fig. 6), both traditional and `ML-KEM`-based configurations exhibit relatively moderate execution times, while those using `HQC` show an approximately threefold time degradation. Across all configurations, the ARM platform consistently achieves lower execution times, resulting in speedup factors ranging from 1.4× to 2.1× for the VPN establishment. This effect is especially pronounced in `hqc256` and its hybrid variant.

An analysis of the TLS handshake time relative to the overall VPN connection establishment reveals the expected trend on both platforms: as the complexity of the KEM increases, the handshake accounts for a larger portion of the total connection time—rising from approximately 40–50% with lightweight schemes like `ML-KEM512` to around 80% with heavier ones like `HQC`—underscoring its increasing impact on overall latency.

In the scenario involving *post-quantum signatures* (Table 9 and Fig. 7), similar trends are observed: `HQC` incurs the highest TLS handshake and OpenVPN connection establishment times, while hybrid and `ML-KEM`-based schemes result in moderate overhead. Across all security levels, ARM consistently outperforms x86, with speedup factors of approximately 1.5-2× for both VPN establishment and TLS handshake execution.

A similar behavior exists with respect to the traditional signature scenario. Both the VPN connection time and TLS handshake duration have increased across nearly all KEMs, especially at higher security levels. This growth stems from the additional computational overhead introduced by post-quantum signature verification and key management during the handshake. Moreover, while the TLS handshake accounts for approximately 50% of the total VPN setup time for traditional primitives and `ML-KEM`, this proportion exceeds 80% in the most demanding `HQC`-based configurations.

In conclusion, traditional and `ML-KEM`-based primitives demonstrate similar behavior across both scenarios, incurring only moderate latency. In contrast, post-quantum schemes based on `HQC` introduce substantially higher latency, directly affecting overall VPN connection performance and underscoring the importance of carefully selecting post-quantum algorithms, particularly for latency-sensitive applications.

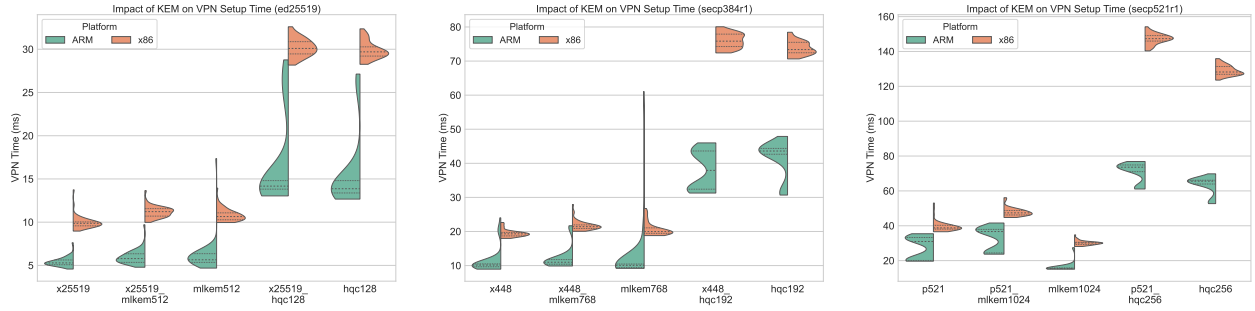
6.3 OpenVPN Handshake Evaluation under realistic conditions

This section illustrates the ability of our framework to simulate realistic network conditions using the Pumba tool. In this study, we selected moderately adverse scenarios, including a fixed delay of 10 ms and a stable loss configuration based on the Gilbert–Elliott (GE) [28, 29] model.

The GE model is a widely adopted two-state Markov process for representing bursty network losses. It al-

Table 8: VPN Connection Establishment (ms) and its embedded TLS handshake with traditional signatures for ARM and x86. "SpeedUp" is the performance advantage of ARM over x86.

Security Level	KEM	ARM						x86						SpeedUp	
		Mean ± SD	Median	IQR	CV	Out. %	TLS Hand.	Mean ± SD	Median	IQR	CV	Out. %	TLS Hand.	VPN	TLS
I	x25519	5.39 ± 0.51	5.29	0.55	0.09	2.04	2.13 ± 0.28	10.18 ± 1.19	9.76	1.10	0.12	8.16	4.06 ± 0.49	1.89×	1.91×
	x25519_mlkem512	5.95 ± 0.82	5.79	1.00	0.14	2.04	2.57 ± 0.55	11.50 ± 1.65	10.93	1.70	0.14	6.12	4.93 ± 0.73	1.93×	1.92×
	mlkem512	6.04 ± 1.75	5.65	1.01	0.29	2.04	2.35 ± 0.37	10.84 ± 1.22	10.32	1.36	0.11	8.16	4.54 ± 0.61	1.79×	1.93×
	x25519_hqc128	16.55 ± 5.14	14.17	0.99	0.31	22.45	12.30 ± 4.39	31.55 ± 3.61	30.27	2.30	0.11	16.33	22.70 ± 2.87	1.91×	1.85×
	hqc128	15.44 ± 4.19	13.87	1.43	0.27	12.24	11.15 ± 3.44	32.66 ± 3.87	31.50	3.69	0.12	10.20	23.31 ± 2.99	2.12×	2.09×
III	x448	11.16 ± 3.47	10.15	0.84	0.31	14.29	5.66 ± 0.48	19.47 ± 1.84	18.89	1.40	0.09	12.24	10.87 ± 1.40	1.74×	1.92×
	x448_mlkem768	12.10 ± 3.21	10.98	1.36	0.27	10.20	6.88 ± 0.60	21.60 ± 2.01	20.95	1.45	0.09	8.16	12.74 ± 1.43	1.78×	1.85×
	mlkem768	11.47 ± 7.56	10.07	1.00	0.66	6.12	5.92 ± 0.60	19.68 ± 1.63	19.12	2.26	0.08	0.00	11.18 ± 1.02	1.72×	1.89×
	x448_hqc192	37.90 ± 5.65	37.93	11.21	0.15	0.00	30.18 ± 5.33	76.49 ± 4.43	75.32	2.70	0.06	10.20	60.53 ± 3.41	2.02×	2.01×
	hqc192	41.46 ± 5.26	43.61	1.71	0.13	24.49	34.49 ± 5.34	74.92 ± 3.68	73.74	3.47	0.05	6.12	59.32 ± 3.06	1.81×	1.72×
V	P-521	26.87 ± 6.51	30.98	13.18	0.24	0.00	14.64 ± 3.17	40.24 ± 4.00	38.94	2.46	0.10	10.20	26.31 ± 2.53	1.50×	1.80×
	p521_mlkem1024	32.39 ± 6.72	36.69	13.50	0.21	0.00	23.12 ± 6.20	48.25 ± 3.79	47.15	2.88	0.08	8.16	32.14 ± 2.72	1.49×	1.39×
	mlkem1024	16.37 ± 2.87	15.46	0.47	0.18	14.29	10.37 ± 0.50	30.68 ± 2.25	30.06	2.08	0.07	6.25	19.43 ± 1.40	1.87×	1.87×
	p521_hqc256	70.98 ± 5.49	73.59	3.84	0.08	24.49	56.88 ± 5.70	148.26 ± 9.02	146.03	4.78	0.06	8.16	119.21 ± 8.83	2.09×	2.10×
	hqc256	63.20 ± 5.05	65.59	2.16	0.08	22.45	52.60 ± 4.99	132.82 ± 11.36	129.41	7.17	0.09	8.16	108.29 ± 11.05	2.10×	2.06×



(a) Level I (b) Level III (c) Level V

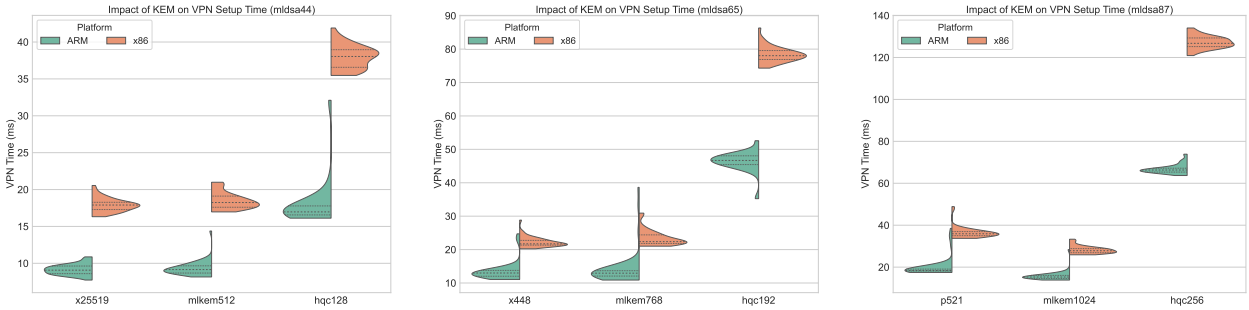
Fig. 6: VPN connection establishment using traditional signatures on ARM and x86

Table 9: VPN Connection Establishment (ms) and its embedded TLS handshake with post-quantum signatures for ARM and x86. "SpeedUp" is the performance advantage of ARM over x86.

Security Level	KEM	ARM						x86						SpeedUp	
		Mean ± SD	Median	IQR	CV	Out. %	TLS Hand.	Mean ± SD	Median	IQR	CV	Out. %	TLS Hand.	VPN	TLS
I	x25519	9.16 ± 0.70	9.07	1.03	0.08	0.00	4.43 ± 0.50	17.89 ± 0.91	17.90	0.98	0.05	4.08	7.86 ± 0.47	1.95×	1.78×
	mlkem512	9.30 ± 0.96	9.15	0.98	0.10	2.04	4.58 ± 0.67	18.50 ± 1.13	18.24	1.51	0.06	0.00	8.29 ± 0.62	1.99×	1.81×
	hqc128	17.84 ± 3.11	16.97	1.22	0.17	8.16	12.35 ± 3.09	38.41 ± 2.49	38.32	2.53	0.06	6.25	25.64 ± 1.58	2.15×	2.08×
III	x448	13.42 ± 2.83	12.99	1.75	0.21	6.12	5.91 ± 0.84	22.25 ± 1.49	21.72	1.41	0.07	6.12	10.14 ± 0.94	1.66×	1.72×
	mlkem768	13.90 ± 5.03	12.94	1.61	0.36	6.12	7.28 ± 3.47	23.43 ± 2.36	22.38	2.57	0.10	6.12	11.07 ± 1.43	1.69×	1.52×
	hqc192	47.31 ± 7.05	46.72	2.63	0.15	10.20	38.48 ± 6.12	78.91 ± 3.61	78.16	2.70	0.05	6.12	59.52 ± 2.84	1.67×	1.55×
V	P-521	19.89 ± 4.58	18.37	0.88	0.23	14.29	11.83 ± 4.16	36.45 ± 2.78	35.92	1.95	0.08	4.08	19.63 ± 1.27	1.83×	1.66×
	mlkem1024	15.67 ± 2.11	15.29	1.45	0.13	4.08	7.83 ± 0.79	28.22 ± 1.78	27.83	1.95	0.06	6.12	13.55 ± 0.87	1.80×	1.73×
	hqc256	66.83 ± 2.18	66.33	1.85	0.03	12.24	55.27 ± 1.94	128.66 ± 7.65	127.41	4.09	0.06	4.08	101.18 ± 6.88	1.93×	1.83×

ternates between a *Good* state, characterized by a low packet loss probability p_g, and a *Bad* state, in which the loss probability is substantially higher, p_b. Transitions between states occur with fixed probabilities, h and k, allowing the model to capture temporal correlations in loss patterns that are not reproduced by simple i.i.d. loss models. This makes the GE model particularly suitable for evaluating the robustness of cryptographic handshakes and transport protocols under realistic and time-correlated impairment conditions.

For our experiments, we employed the *Stable GE-model* configuration, defined by $p_g = 10\%$, $p_b = 50\%$, $h = 70\%$, and $k = 10\%$. This configuration provides a controlled yet realistic environment in which network impairments occur in short bursts rather than independently.

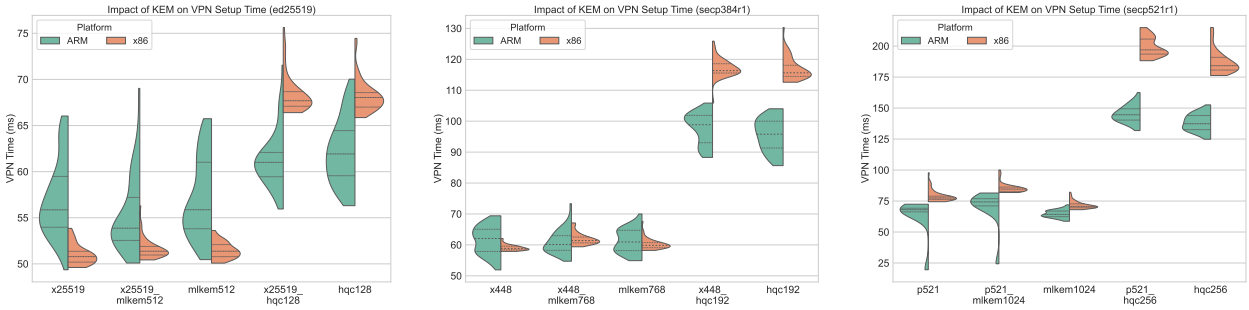


(a) Level I (b) Level III (c) Level V

Fig. 7: VPN connection establishment using post-quantum signatures on ARM and x86

Table 10: VPN Connection Establishment (ms) and its embedded TLS handshake with traditional signatures for ARM and x86, measured under 10 ms network delay. "SpeedUp" is the performance advantage of ARM over x86.

Security Level	KEM	ARM						x86						SpeedUp	
		Mean ± SD	Median	IQR	CV	Out. %	TLS Hand.	Mean ± SD	Median	IQR	CV	Out. %	TLS Hand.	VPN	TLS
I	x25519	57.02 ± 4.10	55.86	5.51	0.07	0.00	14.62 ± 1.63	50.99 ± 1.10	50.79	1.17	0.02	6.12	13.25± 0.35	0.89×	0.91×
	x25519_mlkem512	55.30 ± 4.08	53.87	4.66	0.07	4.08	14.45 ± 1.41	51.63 ± 1.08	51.37	0.93	0.02	6.12	13.27 ± 0.37	0.93×	0.92×
	mlkem512	57.14 ± 4.31	55.87	7.21	0.08	0.00	14.69 ± 1.65	51.46 ± 0.89	51.37	1.31	0.02	0.00	13.00 ± 0.29	0.90×	0.89×
	x25519_hqc128	61.37 ± 3.08	61.01	2.63	0.05	10.20	15.67 ± 0.95	68.92 ± 3.00	67.78	2.07	0.04	12.24	20.78 ± 1.69	1.12×	1.33×
	hqc128	62.18 ± 3.53	61.92	4.89	0.06	0.00	15.41 ± 0.49	68.54 ± 2.52	68.09	1.73	0.04	8.16	21.26 ± 3.53	1.10×	1.38×
III	x448	61.45 ± 4.49	62.07	7.17	0.07	0.00	16.47 ± 2.67	59.07 ± 0.92	58.87	1.20	0.02	4.26	15.26 ± 0.36	0.96×	0.93×
	x448_mlkem768	60.94 ± 3.79	60.12	4.75	0.06	2.04	14.15 ± 0.80	61.95 ± 1.93	61.40	1.88	0.03	8.16	15.82 ± 1.03	1.02×	1.12×
	mlkem768	61.17 ± 3.84	60.95	6.55	0.06	0.00	14.33 ± 1.10	60.25 ± 1.71	59.85	1.63	0.03	6.25	15.17 ± 0.60	0.99×	1.06×
	x448_hqc192	97.61 ± 5.03	98.86	8.83	0.05	0.00	40.38 ± 2.73	117.95 ± 4.81	117.05	3.63	0.04	8.16	66.29 ± 10.60	1.21×	1.64×
	hqc192	95.54 ± 5.26	95.78	8.67	0.06	0.00	39.20 ± 2.67	118.28 ± 6.08	115.96	5.70	0.05	8.16	62.82 ± 11.75	1.24×	1.60×
V	P-521	63.16 ± 14.84	68.05	2.94	0.23	10.20	19.88 ± 4.14	79.00 ±4.67	77.49	2.57	0.06	16.33	25.97 ± 2.39	1.25×	1.31×
	p521_mlkem1024	70.60 ± 14.30	74.29	5.81	0.20	8.16	21.43 ± 4.10	85.34 ± 3.92	84.66	3.49	0.05	8.16	27.00 ± 2.35	1.21×	1.26×
	mlkem1024	64.65 ± 3.00	64.24	4.54	0.05	0.00	14.89 ± 0.41	71.67 ± 3.01	70.67	3.13	0.04	6.12	17.76 ± 1.09	1.11×	1.19×
	p521_hqc256	145.16 ± 6.83	144.60	9.15	0.05	0.00	82.46 ± 4.42	202.18 ± 11.32	197.49	15.04	0.06	4.17	136.92 ± 9.87	1.39×	1.66×
	hqc256	138.18 ± 6.76	137.38	11.38	0.05	0.00	76.57 ± 4.21	186.90 ± 9.26	184.20	10.25	0.05	6.25	122.92 ± 8.89	1.35×	1.61×



(a) Level I (b) Level III (c) Level V

Fig. 8: VPN connection establishment with 10ms delays using traditional signatures on ARM and x86

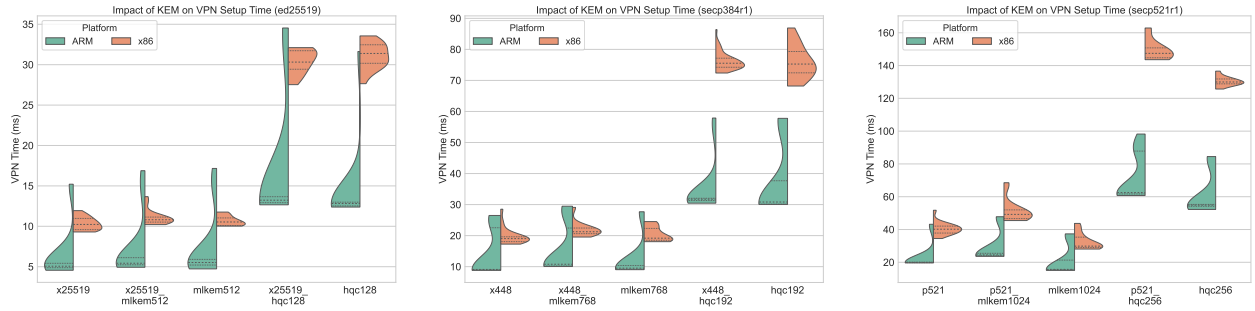
Under delay-dominated conditions, network latency becomes the primary contributor to the overall handshake time, thereby reducing the relative impact of cryptographic complexity across security levels and platforms. For the selected delay of 10 ms, the ARM platform remains advantageous for `HQC`-based configurations. However,this advantage progressively diminishes as the network delay increases.

In the simulated loss-based scenario, the execution time increases slightly, with a more noticeable impact on the standard deviation and the percentage of outliers. However, the speedup of the ARM platform over x86 remains similar to that observed under ideal network conditions.

In summary, we observe that simulating network conditions significantly affects the statistical measure-

Table 11: VPN Connection Establishment (ms) and its embedded TLS handshake with traditional signatures for ARM and x86, measured in a stable-loss network. "SpeedUp" is the performance advantage of ARM over x86.

Security Level	KEM	ARM						x86						SpeedUp	
		Mean ± SD	Median	IQR	CV	Out. %	TLS Hand.	Mean ± SD	Median	IQR	CV	Out. %	TLS Hand.	VPN	TLS
I	x25519	6.38 ± 3.33	5.06	0.54	0.52	23.53	2.79 ± 1.53	10.34 ± 0.82	10.23	1.38	0.08	0.00	3.00 ± 0.32	1.62×	1.07×
	x25519_mlkem512	7.09 ± 3.78	5.43	0.88	0.53	21.05	1.96 ± 1.03	11.06 ±0.92	10.81	0.62	0.08	11.76	3.15 ±0.43	1.56×	1.60×
	mlkem512	7.14 ± 3.92	5.52	0.70	0.55	21.05	2.40 ± 1.34	10.67 ± 0.64	10.54	0.94	0.06	0.00	2.73 ± 0.28	1.49×	1.14×
	x25519_hqc128	16.65 ± 8.07	13.23	0.74	0.48	16.67	9.96 ± 6.81	31.32 ± 3.47	30.88	2.21	0.11	8.33	14.61 ± 6.07	1.88×	1.47×
	hqc128	14.05 ± 4.88	12.82	0.46	0.35	13.33	9.17 ± 5.79	31.34 ± 1.87	31.44	2.53	0.06	0.00	15.38 ± 5.90	2.23×	1.68×
III	x448	14.03 ± 7.80	9.10	13.59	0.56	0.00	4.69 ± 2.68	19.70 ± 2.66	19.03	1.56	0.14	15.00	5.53 ± 1.15	1.40×	1.18×
	x448_mlkem768	15.44 ± 8.16	10.78	12.23	0.53	0.00	5.13 ± 3.38	21.93 ± 2.40	21.25	1.81	0.11	7.69	8.01 ± 4.52	1.42×	1.56×
	mlkem768	12.11 ± 6.54	9.51	1.20	0.54	14.29	5.28 ± 4.15	20.45 ± 2.47	19.17	3.83	0.12	0.00	5.03 ± 0.99	1.69×	1.14×
	x448_hqc192	34.54 ± 8.17	31.74	0.73	0.24	11.76	18.87 ± 9.11	77.08 ± 5.26	75.55	3.02	0.07	10.20	31.58 ± 7.54	2.23×	1.67×
	hqc192	37.15 ± 11.84	30.82	7.56	0.32	25.00	18.93 ± 9.39	75.74 ± 5.55	75.27	6.87	0.07	0.00	36.31 ± 11.39	2.04×	1.92×
V	P-521	22.76 ± 7.72	19.92	0.32	0.34	18.75	12.59 ± 6.73	40.50 ± 3.81	40.17	4.23	0.09	5.56	16.16 ± 2.32	1.78×	1.28×
	p521_mlkem1024	29.26 ± 9.78	24.51	1.43	0.33	21.43	14.52 ± 9.55	51.32 ± 7.02	49.22	5.31	0.14	13.33	20.68 ± 8.96	1.75×	1.42×
	mlkem1024	20.84 ± 9.43	15.68	5.86	0.45	25.00	8.45 ± 5.94	32.45 ± 5.04	29.86	6.27	0.16	0.00	10.05 ± 7.20	1.56×	1.19×
	p521_hqc256	70.97 ± 14.10	62.52	26.14	0.20	0.00	35.70 ± 15.54	154.85 ± 12.70	150.05	17.44	0.08	0.00	67.62 ± 13.38	2.18×	1.89×
	hqc256	60.99 ± 12.74	54.89	1.13	0.21	30.77	31.20 ± 12.16	130.31 ± 2.79	129.98	2.96	0.02	8.33	54.21 ± 6.42	2.14×	1.74×



(a) Level I (b) Level III (c) Level V

Fig. 9: VPN connection establishment under stable packet-loss conditions using traditional signatures on ARM and x86

ments, equalizing the execution time across all configurations; nevertheless, HQC-based configurations remain the worst-performing ones.

7 Multi-Layer Performance Discussion

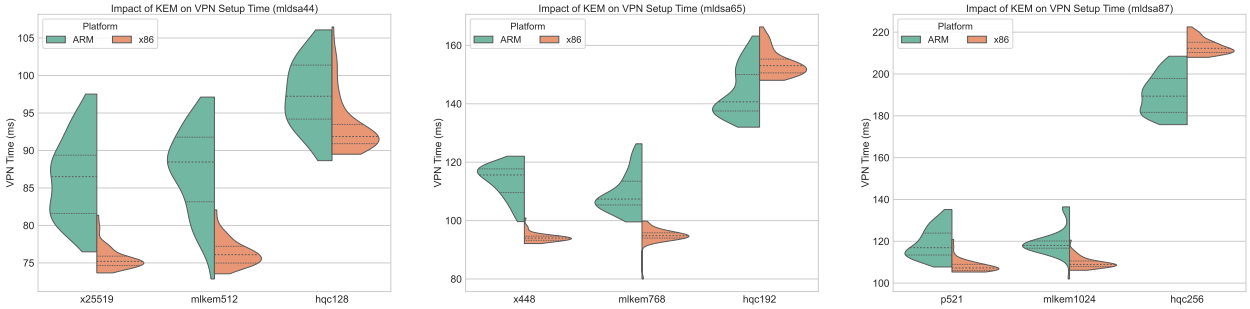
The experimental results reveal that performance trends vary significantly across system layers, highlighting the importance of multi-layer evaluation when assessing post-quantum cryptographic deployments.

- **Primitive layer:** ARM consistently outperforms x86 for most cryptographic primitives. The main exceptions are `ML-KEM` and `ML-DSA`, which benefit from architecture-specific optimizations on x86 under the default `liboqs` configuration.
- **TLS handshake layer:** When cryptographic primitives are integrated into TLS, the x86 advantage observed for `ML-KEM` and `ML-DSA` is largely mitigated. ARM achieves lower end-to-end handshake latency, indicating that the benefits of primitive-level optimizations are overshadowed by protocol processing and other non-cryptographic overheads.
- **VPN layer:** In VPN scenarios, platform differences are further reduced. Under stable network conditions, ARM retains a modest advantage, while under injected delay the performance gap between architectures becomes negligible, showing that network effects dominate connection establishment time.
- **Network conditions:** Network delay and packet loss amplify the contribution of protocol and transport overheads relative to cryptographic computation, reducing the relevance of isolated primitive benchmarks for real-world deployments.

Overall, these results demonstrate that conclusions drawn from isolated cryptographic benchmarks do not necessarily translate to protocol- or application-level performance, reinforcing the need for holistic, multi-layer evaluation of post-quantum cryptographic systems.

Table 12: VPN Connection Establishment (ms) and its embedded TLS handshake with post-quantum signatures for ARM and x86, measured under 10 ms network delays. "SpeedUp" is the performance advantage of ARM over x86.

Security Level	KEM	ARM						x86						SpeedUp	
		Mean ± SD	Median	IQR	CV	Out. %	TLS Hand.	Mean ± SD	Median	IQR	CV	Out. %	TLS Hand.	VPN	TLS
I	x25519	86.28 ± 5.38	86.51	7.77	0.06	0.00	14.59 ± 1.84	75.64 ± 1.52	75.21	1.25	0.02	6.12	12.86 ± 0.38	0.88×	0.88×
	mlkem512	87.24 ± 5.61	88.46	8.61	0.06	0.00	14.67 ± 1.82	76.27 ± 1.76	76.10	2.23	0.02	2.04	13.06 ± 0.33	0.87×	0.89×
	hqc128	97.99 ± 5.25	97.40	7.42	0.05	0.00	15.52 ± 0.46	93.57 ± 4.30	91.86	3.39	0.05	12.24	21.47 ± 2.90	0.95×	1.38×
III	x448	113.38 ± 5.68	115.62	8.10	0.05	0.00	15.44 ± 1.82	94.19 ± 1.57	94.00	1.43	0.02	6.12	13.45 ± 0.44	0.83×	0.87×
	mlkem768	109.40 ± 6.74	107.36	8.13	0.06	2.04	13.65 ± 1.23	94.88 ± 2.70	94.82	1.82	0.03	8.16	13.17 ± 0.34	0.87×	0.96×
	hqc192	144.07 ± 8.48	140.66	12.50	0.06	0.00	37.42 ± 2.02	155.24 ± 7.48	153.30	6.97	0.05	6.12	59.16 ± 12.17	1.08×	1.58×
V	P-521	119.01 ± 7.18	116.95	10.50	0.06	0.00	19.68 ± 1.84	108.19 ± 3.02	107.27	2.88	0.03	6.12	20.92 ± 1.14	0.91×	1.06×
	mlkem1024	119.67 ± 6.57	118.03	3.45	0.05	14.29	13.12 ± 0.94	109.71 ± 2.71	108.93	2.68	0.02	6.12	13.17 ± 0.51	0.92×	1.00×
	hqc256	190.65 ± 9.64	189.40	16.23	0.05	0.00	77.82 ± 3.64	215.25 ± 9.32	212.56	5.20	0.04	6.12	116.31 ± 7.84	1.13×	1.49×



(a) Level I (b) Level III (c) Level V

Fig. 10: VPN connection establishment with 10ms delays using post-quantum signatures on ARM and x86

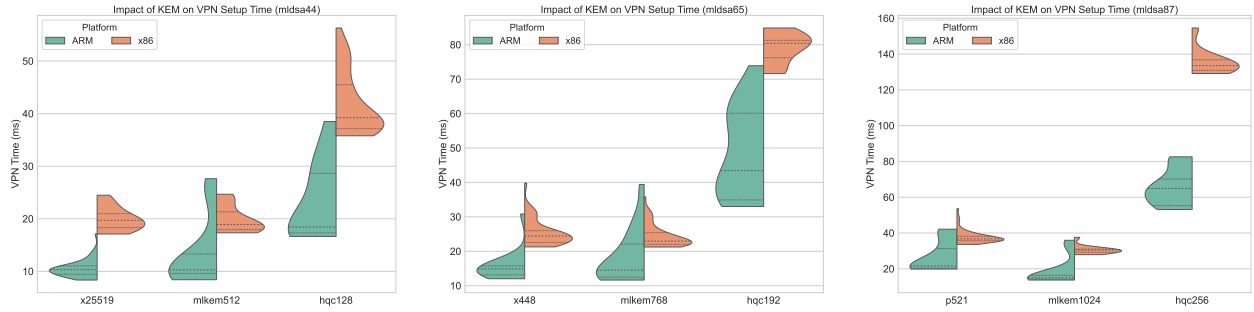
Table 13: VPN Connection Establishment (ms) and its embedded TLS handshake with post-quantum signatures for ARM and x86, measured in a stable-loss network. "SpeedUp" is the performance advantage of ARM over x86.

Security Level	KEM	ARM						x86						SpeedUp	
		Mean ± SD	Median	IQR	CV	Out. %	TLS Hand.	Mean ± SD	Median	IQR	CV	Out. %	TLS Hand.	VPN	TLS
I	x25519	10.50 ± 1.61	10.34	1.60	0.15	6.12	1.96 ± 0.62	19.90 ± 2.32	19.71	2.65	0.12	0.00	3.10 ± 0.63	1.90×	1.43×
	mlkem512	13.45 ± 6.69	10.28	3.72	0.50	18.75	3.23 ± 2.02	19.92 ± 2.68	18.90	3.37	0.13	0.00	3.10 ± 0.63	1.48×	0.96×
	hqc128	23.11 ± 7.72	18.44	11.34	0.33	0.00	8.58 ± 4.33	42.45 ± 6.44	39.26	8.76	0.15	0.00	12.86 ± 5.08	1.84×	1.50×
III	x448	15.60 ± 4.46	14.90	2.62	0.29	6.25	3.57 ± 1.77	25.37 ± 3.89	24.45	3.42	0.15	10.00	3.71 ± 0.80	1.63×	1.04×
	mlkem768	18.14 ± 7.85	14.53	9.63	0.43	5.88	3.75 ± 1.76	24.36 ± 3.32	22.96	3.38	0.14	10.42	3.18 ± 0.61	1.34×	0.85×
	hqc192	48.30 ± 13.54	43.43	25.17	0.28	0.00	20.36 ± 10.59	80.41 ± 5.30	81.03	6.39	0.07	0.00	35.40 ± 13.09	1.66×	1.74×
V	P-521	27.11 ± 9.05	21.53	10.97	0.33	0.00	10.60 ± 5.35	37.81 ± 3.78	36.71	2.44	0.10	8.16	10.96 ± 1.10	1.39×	1.03×
	mlkem1024	18.47 ± 7.85	15.01	2.16	0.43	18.75	4.02 ± 2.33	30.48 ± 2.43	30.45	1.78	0.08	7.69	3.58 ± 1.19	1.65×	0.89×
	hqc256	65.75 ± 10.77	64.90	14.85	0.16	0.00	30.32 ± 11.06	152.76 ± 52.85	134.96	10.95	0.35	16.67	51.81 ± 9.72	2.32×	1.71×

8 Conclusions and Future Work

This work presents a multi-layer performance evaluation of post-quantum cryptographic algorithms, analyzing their impact at several parts of secure communication systems—from standalone cryptographic primitives to secure transport protocol and finally applications—on both x86 and ARM architectures.

Our results indicate that both traditional and post-quantum `ML-KEM` standard variants offer stable and efficient performance across all levels of the communication stack, representing a favorable trade-off between security and efficiency. In contrast, primitives based on the `HQC` scheme—recently proposed for standardization as an alternative to lattice-based primitives—incur substantial overhead, particularly at higher security levels. This overhead propagates through the protocol stack, significantly increasing handshake durations and dominating the total connection setup time in complete protocol deployments under ideal and realistic scenarios.Consequently, `HQC`-based configurations may be impractical for latency sensitive applications, whereas



(a) Level I (b) Level III (c) Level V

Fig. 11: VPN connection establishment under stable packet-loss conditions using traditional signatures on ARM and x86

`ML-KEM` offers a more deployment ready performance profile.

Interestingly, although `ML-KEM` and `ML-DSA` standalone primitives exhibit better performance on x86 compared to ARM, this advantage dissipates at the protocol and application levels. Notably, the ARM platform consistently achieves lower latency across most configurations. These findings suggest that ARM's architecture may be better suited for post-quantum cryptographic workloads in specific networking scenarios.

Consequently, the transition to quantum-safe systems must consider not only the cryptographic strength and efficiency of standalone primitives, but also the performance bottlenecks that emerge when these primitives are integrated across various layers of the protocol stack—particularly in latency-sensitive or resource-constrained environments.

An interesting direction for future work is to extend this multi-layer evaluation to other applications and protocols that rely on TLS for authentication and key exchange. Although this study focused on a VPN-based use case, many widely deployed systems—such as secure messaging platforms, IoT frameworks, cloud service APIs, and microservice architectures—also depend on TLS as a foundational security layer. Assessing the impact of post-quantum cryptographic primitives in these varied contexts would offer a broader understanding of the practical implications of the transition to quantum-safe security, particularly with respect to latency, resource consumption, and scalability.

Author Contributions All authors have contributed equally.

Funding This work has been partially supported by grant PID2022-139268OB-I00 (SecAI project), funded by MICIU/AEI/10.13039/501100011033 and by ERDF/EU. Funding for open access charge: Universidad de Málaga / CBUA. The authors are also grateful to Jesús Rodríguez for his technical support.

Data Availability All data used in the research is publicly available on the repository `https://github.com/montenegro-montes/TLS-VPN`

References

1. P. Shor, Algorithms for quantum computation: discrete logarithms and factoring, in: Proceedings 35th Annual Symposium on Foundations of Computer Science, 1994, pp. 124–134. doi:10.1109/SFCS.1994.365700.
2. NIST CSRC, Post-quantum cryptography, `https://csrc.nist.gov/projects/post-quantum-cryptography`, Accessed: 2025-07 (2025).
3. D. Moody, R. Perlner, A. Regenscheid, A. Robinson, D. Cooper, Transition to post-quantum cryptography, Tech. Rep. NIST IR 8547 ipd, National Institute of Standards and Technology (November 2024). doi:10.6028/NIST.IR.8547.ipd.
4. D. Adrian, E. Stark, P. Francírek, R. Dickson, Tldr.fail, Accessed: 2024-10 (2024). URL `https://tldr.fail/`
5. W. Beullens, Breaking rainbow takes a weekend on a laptop, in: Y. Dodis, T. Shrimpton (Eds.), Advances in Cryptology – CRYPTO 2022, Springer Nature Switzerland, Cham, 2022, pp. 464–479.
6. W. Castryck, T. Decru, An efficient key recovery attack on sidh, in: C. Hazay, M. Stam (Eds.), Advances in Cryptology – EUROCRYPT 2023, Springer Nature Switzerland, Cham, 2023, pp. 423–447.
7. M. Kumar, Post-quantum cryptography algorithm's standardization and perfor-

mance analysis, Array 15 (2022) 100242. doi:10.1016/j.array.2022.100242.
8. C. A. Roma, C.-E. A. Tai, M. A. Hasan, Energy efficiency analysis of post-quantum cryptographic algorithms, IEEE Access 9 (2021) 71295–71317.
9. K.-A. Shim, On the suitability of post-quantum signature schemes for internet of things, IEEE Internet of Things Journal 11 (6) (2024) 10648–10665. doi:10.1109/JIOT.2023.3327400.
10. V. B. Dang, F. Farahmand, M. Andrzejczak, K. Gaj, Implementing and benchmarking three lattice-based post-quantum cryptography algorithms using software/hardware codesign, in: 2019 International Conference on Field-Programmable Technology (ICFPT), 2019, pp. 206–214. doi:10.1109/ICFPT47387.2019.00032.
11. A. Aikata, A. C. Mert, D. Jacquemin, A. Das, D. Matthews, S. Ghosh, S. S. Roy, A unified cryptoprocessor for lattice-based signature and key-exchange, IEEE Transactions on Computers 72 (6) (2023) 1568–1580. doi:10.1109/TC.2022.3215064.
12. D. Sikeridis, P. Kampanakis, M. Devetsikiotis, Post-quantum authentication in TLS 1.3: A performance study, in: Network and Distributed Systems Security (NDSS) Symposium, 2020, pp. 1–16. doi:10.14722/ndss.2020.24203.
13. R. Döring, M. Geitz, Post-quantum cryptography in use: Empirical analysis of the TLS handshake performance, in: NOMS 2022-2022 IEEE/IFIP Network Operations and Management Symposium, 2022, pp. 1–5. doi:10.1109/NOMS54207.2022.9789913.
14. J. A. Montenegro, R. Rios, J. Lopez-Cerezo, A performance evaluation framework for post-quantum tls, Future Generation Computer Systems 175 (2026) 108062. doi:https://doi.org/10.1016/j.future.2025.108062.
URL https://www.sciencedirect.com/science/article/pii/S0167739X25003577
15. M. Sosnowski, F. Wiedner, E. Hauser, L. Steger, D. Schoinianakis, S. Gallenmüller, G. Carle, The performance of post-quantum TLS 1.3, in: Proc. International Conference on emerging Networking EXperiments and Technologies (CoNEXT), 2023, pp. 19,27. doi:10.1145/3624354.3630585.
16. S. Farooq, A. Altaf, F. Iqbal, E. B. Thompson, D. L. R. Vargas, I. d. l. T. Díez, I. Ashraf, Resilience optimization of post-quantum cryptography key encapsulation algorithms, Sensors 23 (12) (2023). doi:10.3390/s23125379.
17. J. Zheng, H. Zhu, Y. Dong, Z. Song, Z. Zhang, Y. Yang, Y. Zhao, Faster post-quantum TLS 1.3 based on ML-KEM: Implementation and assessment, in: J. Garcia-Alfaro, R. Kozik, M. Choraś, S. Katsikas (Eds.), Computer Security – ESORICS 2024, Springer Nature Switzerland, Cham, 2024, pp. 123–143.
18. M. van Heesch, N. van Adrichem, T. Attema, T. Veugen, Towards quantum-safe VPNs and internet, Cryptology ePrint Archive, Paper 2019/1277 (2019).
URL https://eprint.iacr.org/2019/1277
19. M. Saad, W. Ahmed, M. M. Khan, Design and analysis of quantum safe virtual private network, in: 2024 4th International Conference on Innovations in Computer Science (ICONICS), 2024, pp. 1–8. doi:10.1109/ICONICS64289.2024.10824365.
20. C. Paquin, K. Easterbrook, K. Kane, B. LaMacchia, G. Zaverucha, Post-quantum cryptography vpn (2020).
URL https://www.microsoft.com/en-us/research/project/post-quantum-crypto-vpn/
21. S. Bae, Y. Chang, H. Park, M. Kim, Y. Shin, A performance evaluation of ipsec with post-quantum cryptography, in: S.-H. Seo, H. Seo (Eds.), Information Security and Cryptology – ICISC 2022, Springer Nature Switzerland, Cham, 2023, pp. 249–266.
22. Y. R. Mathilde Raynal, Aymeric Genet, Pq-wireguard: we did it again (2021).
URL https://csrc.nist.gov/Presentations/2021/pq-wireguard-we-did-it-again
23. H. Shim, B. Kang, H. Im, D. Jeon, S.-m. Kim, Virtual private network (vpn) with post-quantum cryptography (pqc), in: 2024 15th International Conference on Information and Communication Technology Convergence (ICTC), 2024, pp. 1009–1011. doi:10.1109/ICTC62082.2024.10827179.
24. D. A. Lilly, E. B. Barker, Q. H. Dang, Recommendations for discrete logarithm-based cryptography: Elliptic curve domain parameters (March 2023). doi:10.6028/NIST.SP.800-186.
25. J. Zhang, J. Huang, L. Zhao, D. Chen, Ç. K. Koç, ENG25519: Faster TLS 1.3 handshake using optimized X25519 and Ed25519, in: 33rd USENIX Security Symposium (USENIX Security 24), USENIX Association, Philadelphia, PA, 2024, pp. 6381–6398.
URL https://www.usenix.org/conference/usenixsecurity24/presentation/zhang-jipeng
26. Internet Engineering Task Force (IETF), The transport layer security (TLS) protocol version 1.3, https://www.rfc-editor.org/rfc/rfc8446, Accessed: 2024-09 (2018).
27. OpenVPN Community, Openvpn github repository, https://github.com/OpenVPN/openvpn/,

accessed: 2025-06-30 (2025).
28. E. N. Gilbert, Capacity of a burst-noise channel, Bell System Technical Journal 39 (1960) 1253–1265.
29. E. O. Elliott, Estimates of error rates for codes on burst-noise channels, Bell System Technical Journal 42 (1963) 1977–1997.

A Statistical information

This appendix includes additional statistical information about the data presented in Sec.4.

Table A.1: Additional statistical metrics for the comparison of KEM operation execution times on ARM and x86 architectures.

Security Level	Signature Primitive	ARM			x86		
		KeyGen	Encaps	Decaps	KeyGen	Encaps	Decaps
		Med/IQR/Out.%	Med/IQR/Out.%	Med/IQR/Out.%	Med/IQR/Out.%	Med/IQR/Out.%	Med/IQR/Out.%
I	x25519	0.024 / 0.001 / 0.00	0.054 / 0.001 / 6.00	0.030 / 0.001 / 4.00	0.047 / 0.001 / 2.00	0.106 / 0.002 / 0.00	0.053 / 0.000 / 14.00
III	x448	0.150 / 0.006 / 6.00	0.262 / 0.015 / 0.00	0.113 / 0.007 / 0.00	0.303 / 0.003 / 2.00	0.552 / 0.008 / 0.00	0.243 / 0.003 / 4.00
V	P-521	1.308 / 0.052 / 4.00	2.655 / 0.132 / 0.00	1.330 / 0.080 / 2.00	2.822 / 0.029 / 0.00	5.672 / 0.067 / 0.00	2.842 / 0.036 / 2.00
I	x25519_mlkem512	0.044 / 0.001 / 8.00	0.075 / 0.001 / 16.00	0.081 / 0.001 / 20.00	0.070 / 0.001 / 2.00	0.124 / 0.002 / 2.00	0.122 / 0.002 / 2.00
I	mlkem512	0.020 / 0.000 / 12.00	0.022 / 0.000 / 16.00	0.027 / 0.000 / 8.00	0.013 / 0.001 / 0.00	0.012 / 0.000 / 6.00	0.011 / 0.000 / 24.00
III	x448_mlkem768	0.199 / 0.007 / 2.00	0.321 / 0.007 / 12.00	0.323 / 0.007 / 16.00	0.368 / 0.004 / 2.00	0.632 / 0.007 / 2.00	0.632 / 0.009 / 0.00
III	mlkem768	0.034 / 0.002 / 0.00	0.035 / 0.001 / 10.00	0.044 / 0.003 / 4.00	0.021 / 0.000 / 22.00	0.019 / 0.000 / 2.00	0.018 / 0.001 / 0.00
V	p521_mlkem1024	1.889 / 0.065 / 0.00	2.765 / 0.171 / 2.00	1.586 / 0.110 / 0.00	3.970 / 0.049 / 0.00	5.812 / 0.089 / 2.00	3.302 / 0.036 / 2.00
V	mlkem1024	0.051 / 0.004 / 2.00	0.051 / 0.003 / 2.00	0.060 / 0.004 / 4.00	0.029 / 0.001 / 0.00	0.027 / 0.000 / 22.00	0.027 / 0.000 / 24.00
I	x25519_hqc128	0.895 / 0.062 / 0.00	1.838 / 0.111 / 0.00	3.032 / 0.118 / 4.00	2.614 / 0.034 / 0.00	5.298 / 0.060 / 0.00	8.351 / 0.150 / 0.00
I	hqc128	0.917 / 0.036 / 4.00	1.824 / 0.074 / 6.00	3.019 / 0.103 / 0.00	2.558 / 0.027 / 0.00	5.176 / 0.067 / 2.00	8.237 / 0.075 / 4.00
III	x448_hqc192	2.720 / 0.178 / 0.00	5.447 / 0.252 / 2.00	8.515 / 0.346 / 4.00	8.127 / 0.091 / 2.00	16.276 / 0.184 / 2.00	24.790 / 0.239 / 0.00
III	hqc192	2.776 / 0.067 / 4.00	5.444 / 0.168 / 0.00	8.719 / 0.319 / 0.00	7.831 / 0.079 / 6.00	15.699 / 0.148 / 2.00	24.225 / 0.363 / 0.00
V	p521_hqc256	6.481 / 0.023 / 10.00	12.104 / 0.044 / 4.00	16.824 / 0.061 / 8.00	18.061 / 0.183 / 2.00	34.123 / 0.277 / 6.00	47.441 / 0.438 / 2.00
V	hqc256	4.693 / 0.020 / 8.00	9.488 / 0.043 / 6.00	15.396 / 0.048 / 8.00	14.303 / 0.215 / 0.00	28.820 / 0.344 / 0.00	44.688 / 0.614 / 0.00

Table A.2: Additional statistical metrics for the comparison of Signature operation execution times on ARM and x86 architectures.

Security Level	Signature Primitive	ARM			x86		
		KeyGen	Sign	Verify	KeyGen	Sign	Verify
		Med/IQR/Out.%	Med/IQR/Out.%	Med/IQR/Out.%	Med/IQR/Out.%	Med/IQR/Out.%	Med/IQR/Out.%
I	ed25519	4.168 / 0.160 / 0.00	0.024 / 0.000 / 8.00	0.076 / 0.001 / 12.00	10.819 / 0.627 / 8.00	0.048 / 0.001 / 0.00	0.156 / 0.003 / 0.00
III	ecdsap384	5.378 / 0.193 / 0.00	0.562 / 0.006 / 2.00	0.470 / 0.008 / 2.00	12.756 / 0.592 / 2.00	1.253 /0.017 / 0.00	1.034/ 0.014 / 0.00
V	ecdsap521	7.034 / 0.207 / 0.00	1.404 / 0.022 / 2.00	1.095 / 0.015 / 6.00	15.772 / 0.585 / 2.00	2.989 / 0.054 / 4.000	2.347 / 0.046 / 2.00
I	mldsa44	4.104 / 0.264 / 4.00	0.234 / 0.003 / 2.00	0.059 / 0.000 / 24.00	10.526 / 0.484 / 6.00	0.110 / 0.001 / 4.00	0.039 / 0.000 / 4.00
III	mldsa65	4.489 / 0.260 / 2.00	0.382 / 0.003 / 10.00	0.096 / 0.001 / 4.00	10.635 / 0.617 / 6.00	0.176 / 0.001 / 4.00	0.066 / 0.001 / 0.00
V	mldsa87	4.512 / 0.164 / 6.00	0.459 / 0.005 / 4.00	0.156 / 0.001 / 12.00	10.892 / 0.678 / 0.00	0.221 / 0.002 / 0.00	0.105 / 0.001 / 0.00