electronics

Article

Integrating and Benchmarking KpqC in TLS/X.509

Minjoo Sim [1], Gyeongju Song [1], Siwoo Eum [1], Minwoo Lee [1], Seyoung Yoon [2], Anubhab Baksi [3] and Hwajeong Seo [2,*]

[1] Department of Information Computer Engineering, Hansung University, Seoul 02876, Republic of Korea
[2] Department of Convergence Security, Hansung University, Seoul 02876, Republic of Korea
[3] Department of Computer Science, Lund University, 22100 Lund, Sweden
* Correspondence: hwajeong@hansung.ac.kr; Tel.: +82-760-8033

Abstract

Advances in quantum computing pose a fundamental threat to classical public-key cryptosystems, including RSA and elliptic-curve cryptography (ECC), which form the foundation for authentication and key exchange in the Transport Layer Security (TLS) protocol. In response to these emerging threats, Korea launched the KpqC (Korea Post-Quantum Cryptography) project in 2021 to design, evaluate, and standardize domestic PQC algorithms. To the best of our knowledge, this is the first systematic evaluation of the finalized Korean PQC algorithms (HAETAE, AIMer, SMAUG-T, NTRU+) within a production-grade TLS/X.509 stack, enabling direct comparison against NIST PQC and ECC baselines. To contextualize KpqC performance, we further compare against NIST-standardized PQC algorithms and classical ECC baselines. Our evaluation examines both static overhead (certificate size) and dynamic overhead (TLS 1.3 handshake latency) across computation-bound (localhost) and network-bound (LAN) scenarios, including embedded device and hybrid TLS configurations. Our results show that KpqC certificates are approximately 4.6–48.8× larger than ECC counterparts and generally exceed NIST PQC sizes. In computation-bound tests, both NIST PQC (ML-KEM) and KpqC hybrids exhibited similar handshake latency increases of approximately 8–9× relative to ECC. In network-bound tests, the difference between the two families was negligible, with relative overhead typically around 30–41%. These findings offer practical guidance for balancing security level, key size, packet size, and latency and support phased PQC migration strategies in real-world TLS deployments.

Keywords: post-quantum cryptography; KpqC algorithms; TLS/X.509 integration

Academic Editor: Hung-Yu Chien

Received: 11 August 2025
Revised: 15 September 2025
Accepted: 17 September 2025
Published: 19 September 2025

Citation: Sim, M.; Song, G.; Eum, S.; Lee, M.; Yoon, S.; Baksi, A.; Seo, H. Integrating and Benchmarking KpqC in TLS/X.509. *Electronics* **2025**, *14*, 3717. https://doi.org/10.3390/electronics14183717

1. Introduction

Transport Layer Security (TLS) is the primary protocol for securing internet communications, providing authentication, confidentiality, and integrity through public-key cryptography. While current deployments predominantly rely on RSA and elliptic-curve cryptography (ECC), other public-key paradigms such as code-based, lattice-based, and multivariate cryptosystems have also been proposed and studied in the literature [1]. Among these schemes, RSA and ECC in particular will be rendered insecure once large-scale quantum computers become practical [2]. This vulnerability arises from Shor's algorithm [3], which enables efficient factorization of large integers and computation of discrete logarithms, directly compromising RSA and ECC, and Grover's algorithm [4], which accelerates exhaustive key search attacks and effectively reduces the security level of symmetric-key systems such as AES.

To address these threats, the U.S. National Institute of Standards and Technology (NIST) finalized four post-quantum cryptography (PQC) standards in June 2022: CRYSTALS-KYBER [5], CRYSTALS-DILITHIUM [6], FALCON [7], and SPHINCS+ [8]. In 2024, NIST added HQC [9] as an additional standardized key-encapsulation mechanism (KEM) and initiated the *Additional Signature Competition* to expand its portfolio of quantum-resistant digital signatures [10].

To address this emerging risk, the Republic of Korea has standardized a suite of post-quantum cryptography (PQC) algorithms collectively referred to as Korean post-quantum cryptography (KpqC) [11]. This set includes two signature schemes, HAETAE [12] and AIMer [13], and two KEMs, SMAUG-T [14] and NTRU+ [15], selected through domestic evaluation to meet national security requirements. Conducting a comprehensive performance assessment of KpqC in real-world TLS/X.509 environments has emerged as a critical step for enhancing both the applicability and trustworthiness of the standard, directly informing feasibility and operational cost evaluations in latency-sensitive and resource-constrained settings.

Migrating from classical cryptography to PQC within X.509-based Public Key Infrastructure (PKI) introduces two primary sources of performance overhead. *Static overhead* stems from significantly larger public keys and signatures, which increase certificate sizes, while *dynamic overhead* results from the higher computational costs of PQC primitives, affecting TLS handshake latency [16–19]. These costs are interdependent: the choice of signature scheme directly influences certificate size, and the selected KEM determines the runtime latency. Therefore, a unified evaluation considering both aspects is necessary to accurately characterize PQC migration trade-offs.

Although prior studies have benchmarked NIST-standardized PQC algorithms, deployment-level performance of Korean PQC (KpqC) schemes in production grade TLS/X.509 settings remains unexplored. This absence of side-by-side comparisons against both NIST PQC and ECC limits the availability of practical guidance for migration planning.

To close this gap, we present the first systematic evaluation of standardized KpqC algorithms—HAETAE, AIMer, SMAUG-T, and NTRU+—within the TLS/X.509 framework, implemented via the Open Quantum Safe (OQS) ecosystem. We integrated these schemes into OQS-enabled OpenSSL, enabling controlled comparisons against NIST PQC and ECC under computation-bound (localhost) and network-bound (LAN) conditions, including embedded-device and hybrid TLS configurations.

The remainder of this paper is organized as follows. Section 2 reviews background, Section 3 describes our integration and experimental setup, Section 4 presents results and analysis, and Section 5 concludes with future research directions.

Contributions

This paper makes the following contributions:

- **First publicly available integration of KpqC within a production-grade TLS stack.** We integrate the standardized Korean post-quantum cryptography (KpqC) schemes HAETAE, AIMer, SMAUG-T, and NTRU+ into the widely deployed OpenSSL library via the Open Quantum Safe (OQS) framework. This implementation supports full TLS 1.3 handshakes, X.509 certificate issuance and validation, and hybrid certificate configurations, enabling reproducible experiments without altering the TLS protocol flow. The source code is publicly released at https://github.com/minjoo97/liboqs_KpqC and https://github.com/minjoo97/oqs-provider_KpqC (accessed on 16 September 2025.), lowering the barrier for independent benchmarking and pilot deployments.

- **Comprehensive and controlled benchmarking of hybrid X.509 certificates.** We conduct a unified, apples-to-apples performance evaluation that measures both static overhead (certificate size) and dynamic overhead (TLS 1.3 handshake latency). The benchmarking covers computation-bound (localhost) and network-bound (LAN) environments, as well as embedded-device scenarios, directly comparing KpqC with NIST PQC standards and classical ECC under identical cryptographic and network conditions.
- **Evidence-based deployment guidelines for PQC migration.** Based on empirical results, we analyze the trade-offs among certificate size, handshake latency, and security level. Our findings show that the substantial computational overhead observed in isolated environments is markedly reduced under realistic network conditions. We present practical migration strategies, including hybrid certificate adoption and algorithm selection frameworks, to balance security objectives with operational performance requirements.

2. Background

2.1. Open Quantum Safe & Liboqs

The Open Quantum Safe (OQS) project aims to facilitate the transition to post-quantum cryptography (PQC) by integrating quantum-resistant algorithms into existing protocols and applications [20]. Its core library, `liboqs`, provides a standardized C API for a wide range of PQC key encapsulation mechanisms (KEMs) and digital signature schemes, while the associated `oqs-provider` enables their use within the OpenSSL 3.x framework for testing and benchmarking [21].

At the time of this research, the OQS ecosystem lacked official support for the algorithms selected in the Korean post-quantum cryptography (KpqC) standardization program. This absence posed a significant barrier to empirical evaluation and hindered direct comparison with finalized NIST standards, motivating the implementation work presented in this paper.

2.2. NIST PQC Standards

Following a multi-year evaluation process, the U.S. National Institute of Standards and Technology (NIST) has standardized a set of post-quantum cryptographic algorithms designed to withstand both classical and quantum adversaries. The selected schemes include the lattice-based CRYSTALS-Kyber (ML-KEM) for key encapsulation and CRYSTALS-Dilithium (ML-DSA) for digital signatures, as well as FALCON for compact signatures and SPHINCS+ (SLH-DSA) as a stateless hash-based alternative. Table 1 summarizes the parameter sizes of the primary KEM and digital signature algorithms standardized by NIST, which serve as the baseline for the comparative evaluation with Korean PQC (KpqC) algorithms in this study.

2.3. X.509 Certificates & PQ Hybrid Certification

X.509 certificates, standardized by ITU-T as part of the X.500 series, define a DER-encoded format for public-key data, issuer/subject names, validity periods, and digital signatures [22]. Extensions allow embedding custom fields—used here to carry KpqC public keys and signatures.

Post-quantum (PQ) hybrid certification combines classical and PQC schemes within a single certificate to ensure continuity and security during migration [23–25]. Common approaches include composite certificates (parallel algorithms, See Figure 1), hybrid signatures (dual-algorithm signing), and chameleon certificates (dynamic field adaptation) [26]. By integrating multiple schemes, hybrid certificates remain secure if one algorithm is broken, offering adaptable security profiles across environments.

Table 1. Parameter sizes (in bytes) of standardized NIST PQC algorithms, including public key, secret key, and ciphertext/signature lengths, used as a baseline for comparative evaluation in this study.

Type	Algorithm	Security Lvl.	Public Key (Bytes)	Secret Key (Bytes)	Ciphertext/Sig. (Bytes)
KEM	ML-KEM-512	1	800	1632	768
	ML-KEM-768	3	1184	2400	1088
	ML-KEM-1024	5	1568	3168	1568
Signature	ML-DSA-44	2	1312	2528	2420
	ML-DSA-65	3	1952	4000	3293
	ML-DSA-87	5	2592	4864	4595
Signature	Falcon-512	1	897	1281	666
	Falcon-1024	5	1793	2305	1280
Signature	SPHINCS+-128s	1	32	64	7856
	SPHINCS+-128f	1	32	64	17,088
	SPHINCS+-192s	3	48	96	16,224
	SPHINCS+-192f	3	48	96	35,664
	SPHINCS+-256s	5	64	128	29,792
	SPHINCS+-256f	5	64	128	49,856

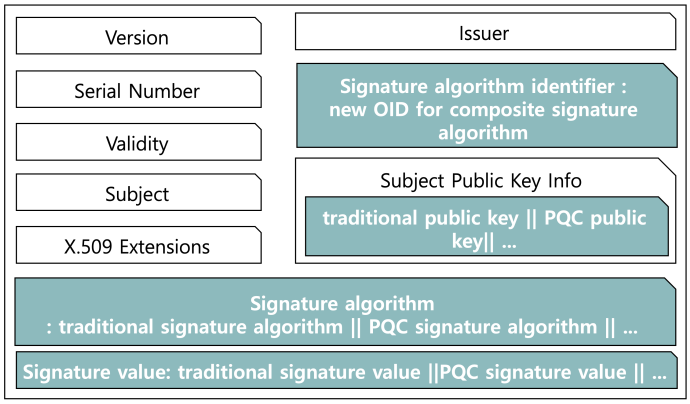

Figure 1. Illustration of composite hybrid certificate structure. Multiple public keys and signatures are serialized under a composite OID, with concatenated values wrapped in Listing 1 ASN.1 SEQUENCEs.

Listing 1. ASN.1 sketch (simplified).

```
SubjectPublicKeyInfo ::= SEQUENCE {
   algorithm         AlgorithmIdentifier { id-alg-composite },
   subjectPublicKey  BIT STRING {
       SEQUENCE {
          AlgorithmIdentifier + EC public key,
          AlgorithmIdentifier + PQC public key
       }
   }
}
```

2.4. KpqC Standardization

Alongside global standardization efforts, Korea launched its national PQC competition (KpqC) in 2022 to establish domestic standards [11]. Initially comprising 16 candidates, the competition advanced eight algorithms to Round 2 in December 2023: four KEMs (NTRU+, PALOMA, REDOG, SMAUG-T), and four digital signature schemes (AIMer, HAETAE, MQ-Sign, NCC-Sign).

On 16 January 2025, the final KpqC standard was announced, selecting two KEMs NTRU+ and SMAUG-T and two digital signature schemes AIMer and HAETAE through a national cryptographic evaluation process to meet domestic security policy requirements. Table 2 summarizes the parameter sizes for these algorithms.

Table 2. Parameter sizes (in bytes) of standardized Korean post-quantum cryptography (KpqC) algorithms, covering public key, secret key, and ciphertext/signature lengths as specified in the final 2025 KpqC standard.

Type	Algorithm	Security Level	Public Key (Bytes)	Secret Key (Bytes)	Ciphertext/Sig. (Bytes)
Key Encapsulation Mechanisms (KEM)					
SMAUG-T	smaug_t1	1	672	832	672
	smaug_t3	3	1088	1312	992
	smaug_t5	5	1440	1728	1376
NTRU+	ntru_plus_kem576	1	864	1760	864
	ntru_plus_kem768	3	1152	2336	1152
	ntru_plus_kem864	3	1296	2624	1296
	ntru_plus_kem1152	5	1728	3488	1728
Digital Signature Algorithms (DSA)					
HAETAE	haetae_2	2	992	1408	1474
	haetae_3	3	1472	2112	2349
	haetae_5	5	2080	2752	2948
AIMer	aimer_128s	1	32	48	4160
	aimer_128f	1	32	48	5888
	aimer_192s	3	48	72	9120
	aimer_192f	3	48	72	13,056
	aimer_256s	5	64	96	17,056
	aimer_256f	5	64	96	25,120

2.4.1. NTRU+

NTRU+ is an enhanced variant of NTRU proposed to overcome structural weaknesses in the original design. It applies the Adaptive Chosen Worst-Case to Average-Case (ACWC) transformation to establish security reductions from worst-case lattice problems, and employs SOTP (Semi-generalized One-Time Pad) message encoding to improve correctness and efficiency. Re-encryption techniques are incorporated to strengthen IND-CPA security and reduce decryption errors. The scheme is instantiated over cyclotomic rings

$$R_q = \mathbb{Z}_q[x]/(x^n - 1),$$

and optimized through Radix-2/3 NTT multiplication, AVX-based parallelism, and lazy modular reduction, providing competitive performance across security levels.

2.4.2. SMAUG-T

SMAUG-T is an efficient lattice-based KEM whose security relies on both Module-LWE (MLWE) and Module-LWR (MLWR). It achieves IND-CCA2 security via a Fujisaki–Okamoto transform. SMAUG-T employs sparse secrets and module structure to improve runtime and reduce sizes. In the TiMER variant (a variant of SMAUG-T), D2 encoding is used to decrease ciphertext and key sizes, with parameter choices tuned to balance decryption failure probability. The design uses rounding operations and sparse secret sampling (e.g., HWTh sampler) rather than "scPBD" sampling. For many parameter sets, ciphertext sizes are smaller compared to Kyber or Saber for similar security levels.

2.4.3. AIMer

AIMer is a signature scheme built from a one-way function (named AIM) and a non-interactive zero-knowledge proof of knowledge (NIZKPoK) system based on the MPC-in-the-Head paradigm. In particular, AIMer uses the BN++ proof system with optimizations, including combining Commit and ExpandTape, domain separation of hash/XOF functions, message pre-hashing, and reduced salt size. The one-way function AIM itself incorporates Mersenne exponentiation S-boxes, linear layers (affine transformations), constants, and tweakable seed components, providing concrete security based on symmetric primitives. The updated version (AIMer v2) includes further improvements in implementation performance and flexibility.

2.4.4. HAETAE

HAETAE is a module-lattice-based digital signature scheme built from Module-LWE and Module-SIS, following the *Fiat–Shamir with Aborts* paradigm. Unlike CRYSTALS-DILITHIUM, HAETAE employs bimodal distributions (inspired by BLISS) for rejection sampling and draws samples from hyperball-uniform distributions rather than hypercubes, which enables more compact representations and improved efficiency. In particular, HAETAE achieves approximately 30–40% smaller signature and key sizes compared to Dilithium at equivalent security levels, while the verification key size is reduced by about 20–25%. The design also incorporates features that facilitate constant-time implementation and improve resilience against side-channel leakage.

2.5. Related Work on KpqC Performance Evaluation

Previous research on post-quantum cryptography has followed three main trajectories, yet none have addressed the practical performance of KpqC in real-world TLS environments. First, while several studies have benchmarked KpqC algorithms at the implementation level, they measure performance in isolation, overlooking the overhead of protocol integration and network communication [27–29]. Second, extensive research on integrating PQC into TLS has focused almost exclusively on NIST-standardized algorithms or candidates from the NIST PQC competition, as these efforts were initiated earlier and thus became the primary focus of implementation studies [18,19]. Finally, prior work on hybrid certificate frameworks has not extended evaluation to KpqC schemes. Kampanakis et al. [17] implemented and exchanged hybrid X.509 certificates in TLS, QUIC, and IKEv2 using selected NIST PQC candidates (e.g., HSS), but did not measure cryptographic computation overhead and evaluated only a limited algorithm set. Chen et al. [16] benchmarked NIST-standard PQC algorithms (Falcon, Dilithium, SPHINCS+) in X.509/TLS integration, but did not implement or assess hybrid certificate deployments in networked environments.

This study fills these gaps by presenting the first comprehensive performance evaluation of KpqC-based hybrid certificates in a production-grade TLS 1.3 implementation, providing empirical data to guide real-world deployment strategies.

3. Proposed Method

This study quantitatively evaluates the static (certificate size) and dynamic (TLS 1.3 handshake latency) overheads introduced by deploying Korean post-quantum cryptography (KpqC) algorithms into production-grade PKI systems. Our methodology comprised two primary phases: (i) extending the Open Quantum Safe (OQS) framework to incorporate four standardized KpqC schemes, and (ii) conducting controlled benchmarking under diverse deployment conditions. The overall workflow is illustrated in Figure 2.

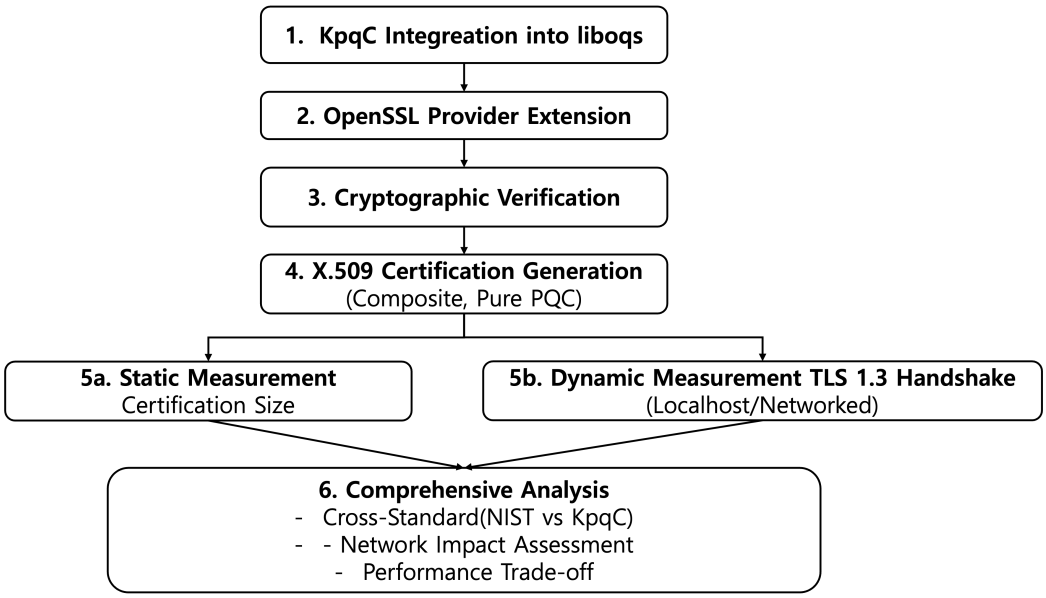

Figure 2. Workflow for evaluating the practical overhead of KpqC in X.509 certificates and TLS 1.3 handshakes.

3.1. Extending the OQS Framework with KpqC Algorithms

To facilitate deployment-level evaluation, we extended the OQS framework by incorporating SMAUG-T, NTRU+, HAETAE, and AIMer into `liboqs` and the `oqs-provider` for OpenSSL 3.x. Rather than a direct code transplant, the process followed three design principles: (i) algorithmic fidelity, (ii) API interoperability, and (iii) performance-conscious implementation. This ensured that KpqC schemes became fully interoperable within the OQS ecosystem and directly comparable to NIST PQC algorithms under identical conditions.

- **API Uniformity:** Each algorithm was encapsulated with thin wrappers adhering to the standardized `liboqs` interface, enabling transparent invocation by OQS-compatible applications. Signature schemes implemented the full API set (`OQS_SIG_keypair`, `OQS_SIG_sign`, `OQS_SIG_verify`), using the constant-time memory cleansing function (`OQS_mem_cleanse`) from liboqs to securely erase private-key material. KEMs implemented (`OQS_KEM_keypair`, `OQS_KEM_encaps`, `OQS_KEM_decaps`) with deterministic randomness for Known Answer Tests (KATs).
- **Modularity and Extensibility:** The build system was extended with per-algorithm compilation flags (e.g., `-DOQS_ENABLE_SIG_haetae_3=ON`), enabling partial builds for resource-constrained environments and simplifying future algorithm updates.
- **Upstream Compatibility:** Code structure and API contracts were preserved to align with upstream OQS development, ensuring minimal maintenance effort and future merge compatibility.

Verification and Testing

KAT vectors from the official KpqC implementations were incorporated into the `liboqs` test suite. Automated tests validated correctness on x86_64 (GCC 13.2, Clang 17) and ARMv8-A (aarch64, GCC 13.2) platforms. For example, the following commands execute algorithm-specific test cases:

```
ctest -R haetae
ctest -R smaug
```

This cross-architecture validation ensured functional reliability across heterogeneous deployment environments.

3.2. Extending the OQS Provider for TLS Integration

While `liboqs` provided the cryptographic primitives, full TLS 1.3 integration required modifications to the `oqs-provider`. The stock provider lacks support for KpqC-specific identifiers, encodings, and dispatch bindings. To enable production-grade testing, the following extensions were implemented:

- **Algorithm Registration:** Added unique identifiers for HAETAE, AIMer, SMAUG-T, and NTRU+ in the provider source, linking them to their respective `liboqs` API calls through dedicated `OSSL_DISPATCH` tables.
- **ASN.1 and OID Support:** Defined and registered new object identifiers (OIDs) within OpenSSL's ASN.1 module to enable DER/PEM encoding, decoding, and hybrid certificate construction.
- **Build and Configuration:** Updated build scripts and `providers.conf` to allow dynamic loading of the extended provider via the `-provider oqsprovider` option.
- **End-to-End Validation:** Verified TLS 1.3 handshakes with KpqC-only and hybrid certificates on both x86_64 and ARMv8 platforms, confirming compatibility with OpenSSL's protocol flow and certificate validation.

These enhancements transformed the OQS–OpenSSL stack into a complete experimental platform for assessing KpqC in realistic PKI workflows.

3.3. Measurement Procedure

Two categories of overhead were evaluated:

3.3.1. Static Overhead: Certificate Size

DER-encoded X.509 certificates were generated for the following:

1. ECC-only (P-256, P-384);
2. PQC-only (HAETAE, AIMer, SMAUG-T, NTRU+);
3. Hybrid ECC+PQC (composite public key and signature).

Certificate sizes were measured using:

```
openssl x509 -in cert.pem -outform DER | wc -c
```

Hybrid certificates followed the composite format described in [24], embedding multiple `SubjectPublicKeyInfo` structures in a single ASN.1 sequence.

3.3.2. Dynamic Overhead: TLS 1.3 Handshake Latency

Handshake latency was measured using OpenSSL's `s_server` and `s_time` utilities:

```
# Server
openssl s_server -accept 8443 -cert server.crt -key server.key \
-tls1_3 -www -provider oqsprovider



# Client (200 sequential TLS 1.3 handshakes)
for i in $(seq 1 200); do
  openssl s_client -connect 127.0.0.1:8443 -tls1_3 -groups \
  smaug_t1 -provider oqsprovider
done
```

Two scenarios were tested:

- **Computation-Bound:** Both endpoints on a MacBook Pro (M1 Pro, 16 GB RAM, macOS 14.4) using `localhost`.

- **Network/Resource-Constrained:** Server deployed on a Raspberry Pi 5 (Cortex-A76, 8 GB RAM, Ubuntu 24.04), with the client running on a MacBook Pro over IEEE 802.11ac LAN. This setup models an *IoT/edge deployment*, where lightweight edge devices terminate TLS sessions with resource-rich cloud clients, thereby reflecting asymmetric computational and networking constraints in real-world PQC adoption scenarios.

For statistical reliability, each scenario was repeated in five independent runs to account for run-to-run variability.

4. Results and Analysis

Building on the implementation described in Section 3, this section presents a comprehensive evaluation of the integrated PQC and hybrid TLS framework. We analyze both *static overhead*, in terms of DER-encoded certificate size, and *dynamic overhead*, measured as TLS 1.3 handshake performance in controlled computational and networked scenarios. Overall, our results show that KpqC certificates are between 11.5× and 48× larger than their ECC counterparts, while handshake latency increases by approximately 8–9× in computation-bound localhost experiments. In contrast, in LAN experiments the relative overhead remains below 40%. To validate these findings, we report dispersion measures including medians, interquartile ranges (IQR), and 95% confidence intervals in Appendix A Table A1.

For all TLS 1.3 experiments, the server authentication certificate was fixed to ECDSA with P-256 (secp256r1) across all configurations. This choice was made to ensure fairness and to isolate the performance impact of the key exchange (KEM), while keeping the signature component constant. As a result, the reported handshake latencies primarily reflect the contribution of the key exchange, rather than variations in signature size or verification cost.

4.1. Experimental Setup

Performance was evaluated in two environments. In the *computational scenario*, both the client and server ran on a single MacBook Pro (M1 Pro, 16 GB RAM) to measure pure cryptographic overhead without network latency. In the *networked scenario*, the client (MacBook Pro) communicated over IEEE 802.11ac Wi-Fi with a Raspberry Pi 5 server (ARM-A76, 8 GB RAM) configured with a static IP. Both systems ran OpenSSL 3.1.2 extended with a modified OQS provider supporting additional Korean post-quantum cryptographic (KpqC) algorithms.

The integration process involved adding new KEM and signature schemes via `generate.py` templates, defining parameter sets and API bindings, and updating provider configuration files to register them in OpenSSL's fetch mechanism. This ensured full compatibility for classical, PQC-only, and hybrid key exchange without modifying TLS 1.3 handshake logic.

4.2. Static Overhead: Certificate Size

X.509 certificates were generated for all tested signature schemes, and their DER-encoded sizes were measured. Table 3 summarizes the results. Classical ECC certificates were the most compact, with secp256r1 at 385 B. Among PQC schemes at Level 1, Falcon512 produced the smallest certificate (1788 B), representing a 4.6× increase over ECC. KpqC signatures (e.g., aimer128s, aimer128f) were substantially larger, up to 16× the ECC baseline.

Table 3. DER-encoded certificate sizes for PQC-only and hybrid signatures.

Family	Algorithm	Level	PQC-Only (B)	Hybrid (B)
NIST Level 1 & 2				
Classical	secp256r1	1.000	385.000	-
NIST PQC	falcon512	1.000	1.788	1.941
KpqC	aimer128s	1.000	4.427	4.582
KpqC	aimer128f	1.000	6.155	6.310
NIST PQC	sphincsshake128fsimple	1.000	17.382	17.536
NIST PQC	mldsa44	2.000	3.977	4.120
KpqC	haetae2	2.000	2.702	2.857
NIST Level 3				
Classical	secp384r1	3.000	447.000	-
NIST PQC	mldsa65	3.000	5.506	5.713
KpqC	haetae3	3.000	4.057	4.276
KpqC	aimer192s	3.000	9.404	9.624
KpqC	aimer192f	3.000	13.340	13.559
NIST Level 5				
Classical	secp521r1	5.000	521.000	-
NIST PQC	mldsa87	5.000	7.464	7.742
NIST PQC	falcon1024	5.000	3.304	3.596
KpqC	haetae5	5.000	5.264	5.554
KpqC	aimer256s	5.000	17.356	17.647
KpqC	aimer256f	5.000	25.420	25.711

Hybrid certificates, combining classical and PQC signatures, exhibited only a marginal size increase over their PQC-only counterparts (typically $< 2\%$), indicating that PQC signature data dominates overall size growth. Figure 3 illustrates the comparison for representative Level 1 and Level 2 algorithms.

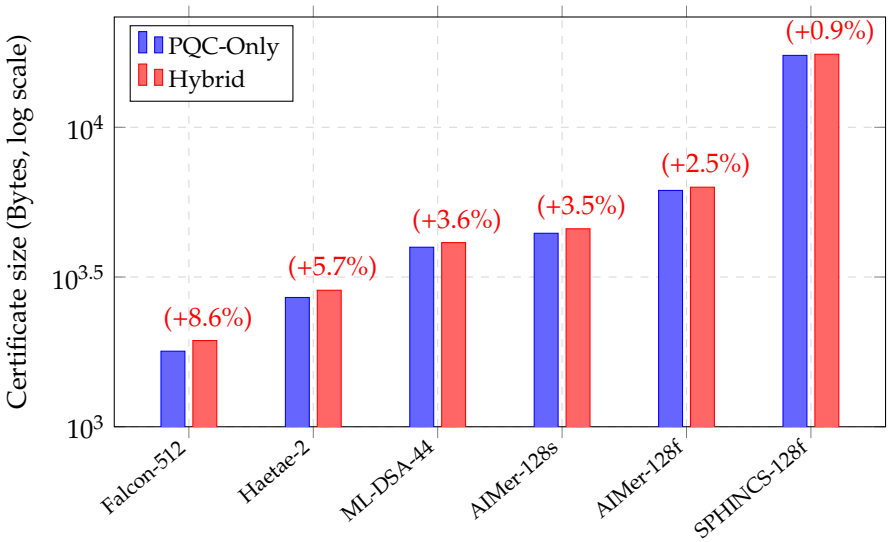

Figure 3. Certificate sizes of PQC-only vs. Hybrid X.509 (log scale). Hybrid overhead remains modest, mostly under 2%, with the maximum at 8.6% (Falcon-512).

4.3. Dynamic Overhead: TLS Handshake Performance

TLS 1.3 handshake times were measured over 200 consecutive connections. In the localhost setup (Table 4), both NIST PQC (ML-KEM) and KpqC hybrids exhibited similar computational cost, with handshake times increasing by approximately 8–9$\times$ relative to

ECC. In the networked scenario (Table 5), the performance gap between the two families was negligible, with relative overhead remaining under 40%. In the networked scenario (Table 5), absolute handshake times increased across all algorithms due to network latency. The relative overhead of PQC and KpqC hybrids dropped significantly (to 30–40%) compared to localhost results, indicating that network delay masks much of the cryptographic cost. KpqC hybrids showed performance comparable to NIST PQC hybrids at the same security level.

Table 4. TLS 1.3 handshake performance (localhost, results over a 200 s measurement window).

Family	Key Exchange	Time (ms)	Throughput (hps)
Level 1			
Classical	secp256r1	5.240	190.700
NIST PQC	secp256r1 + mlkem512	47.970	20.880
KpqC	secp256r1 + smaug_t1	45.000	22.220
KpqC	secp256r1 + ntru_plus_kem576	45.090	22.180
Level 3			
Classical	secp384r1	5.190	192.510
NIST PQC	secp384r1 + mlkem768	48.480	20.630
KpqC	secp384r1 + smaug_t3	46.540	21.490
KpqC	secp384r1 + ntru_plus_kem768	46.060	21.710
KpqC	secp384r1 + ntru_plus_kem864	46.190	21.650
Level 5			
Classical	secp521r1	5.180	193.110
NIST PQC	secp521r1 + mlkem1024	47.920	20.870
KpqC	secp521r1 + smaug_t5	47.150	21.210
KpqC	secp521r1 + ntru_plus_kem1152	46.690	21.420

Table 5. TLS 1.3 handshake performance (MacBook–Raspberry Pi, results over a 200 s measurement window).

Family	Key Exchange	Time (ms)	Overhead
Level 1			
Classical	secp256r1	98.32	-
NIST PQC	secp256r1 + mlkem512	130.14	+32.36 %
KpqC	secp256r1 + smaug_t1	130.15	+32.37 %
KpqC	secp256r1 + ntru_plus_kem576	135.68	+38.00 %
Level 3			
Classical	secp384r1	109.90	-
NIST PQC	secp384r1 + mlkem768	144.75	+31.71 %
KpqC	secp384r1 + smaug_t3	152.04	+38.34 %
KpqC	secp384r1 + ntru_plus_kem768	146.70	+33.48 %
KpqC	secp384r1 + ntru_plus_kem864	150.63	+37.06 %
Level 5			
Classical	secp521r1	125.19	-
NIST PQC	secp521r1 + mlkem1024	167.41	+33.72 %
KpqC	secp521r1 + smaug_t5	176.40	+40.90 %
KpqC	secp521r1 + ntru_plus_kem1152	166.04	+32.63 %

5. Conclusions

This study presented a comprehensive evaluation of static and dynamic overheads when integrating PQC into the X.509/TLS framework and KpqC algorithms with NIST-standardized counterparts. Certificate sizes increased by approximately 4.6× to 48.8×, from Falcon to AIMer. In pure computational scenarios, both ML-KEM and KpqC hybrids exhibited handshake latencies roughly 8–9× higher than ECC baselines. Under realistic network conditions, KpqC and NIST PQC hybrids showed similar performance.

These results underscore a trade-off between bandwidth/storage overhead and computational cost, providing guidance for algorithm selection based on deployment constraints.

Compact NIST PQC schemes may suit bandwidth- or storage-constrained environments, whereas high-capacity systems can adopt KpqC algorithms when their cryptographic properties are desirable.

Future work will explore alternative hybrid frameworks (e.g., chameleon-style designs) and assess additional metrics such as certificate path validation time. Benchmarking in wide-area networks with varying latency, jitter, and packet loss will further clarify PQC deployment challenges in real-world conditions.

Author Contributions: Software, M.S., S.E. and M.L.; Writing—original draft, M.S.; Writing—review & editing, G.S., S.Y., A.B. and H.S.; Supervision, H.S. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by Institute for Information & communications Technology Promotion (IITP) grant funded by the Korea government (MSIT) (No. 2018-0-00264, Research on Blockchain Security Technology for IoT Services, 20%) and this work was supported by Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No. RS-2025-02306395, Development and Demonstration of PQC-Based Joint Certificate PKI Technology, 50%) and this work was supported by the Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No. RS-2025-25394739, Development of Security Enhancement Technology for Industrial Control Systems Based on S/HBOM Supply Chain Protection, 30%).

Data Availability Statement: The data presented in this study are available on request from the corresponding author.

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

Appendix A

Appendix A.1. Detailed Handshake Statistics

Table A1. Summary statistics (Mean, Median, IQR, 95% CI) of TLS 1.3 handshake latency (local). Grouped by security level (1, 3, 5), ordered as ECC baseline → NIST PQC → KpqC variants.

Algorithm	Mode	Mean (s)	Median (s)	IQR	95% CI Low	95% CI High	N
secp256r1	fixed	0.03681	0.03647	0.00048	0.03666	0.03695	200
p256_mlkem512	fixed	0.05067	0.04974	0.00181	0.04981	0.05153	200
p256_smaug_t1	fixed	0.04896	0.04846	0.00071	0.04867	0.04926	200
p256_ntru_plus_kem576	fixed	0.04875	0.04835	0.00052	0.04850	0.04899	200
secp384r1	fixed	0.03890	0.03832	0.00130	0.03849	0.03931	200
p384_mlkem768	fixed	0.04939	0.04913	0.00060	0.04921	0.04958	200
p384_smaug_t3	fixed	0.05937	0.05672	0.00302	0.05767	0.06106	200
p384_ntru_plus_kem768	fixed	0.04936	0.04902	0.00068	0.04914	0.04958	200
p384_ntru_plus_kem864	fixed	0.05000	0.04957	0.00097	0.04971	0.05029	200
secp521r1	fixed	0.03771	0.03748	0.00064	0.03758	0.03783	200
p521_mlkem1024	fixed	0.04946	0.04917	0.00050	0.04934	0.04958	200
p521_smaug_t5	fixed	0.04975	0.04938	0.00068	0.04955	0.04995	200
p521_ntru_plus_kem1152	fixed	0.04942	0.04915	0.00060	0.04913	0.04971	200
secp256r1	time	0.03678	0.03655	0.00125	0.03669	0.03687	3232
p256_mlkem512	time	0.04941	0.04875	0.00244	0.04917	0.04965	2109
p256_smaug_t1	time	0.04952	0.04857	0.00373	0.04925	0.04979	2108
p256_ntru_plus_kem576	time	0.04877	0.04843	0.00179	0.04860	0.04894	2104
secp384r1	time	0.03887	0.03848	0.00171	0.03872	0.03902	3094
p384_mlkem768	time	0.04942	0.04910	0.00170	0.04925	0.04959	2103
p384_smaug_t3	time	0.05908	0.05686	0.00999	0.05836	0.05981	2049
p384_ntru_plus_kem768	time	0.04939	0.04905	0.00164	0.04922	0.04956	2107
p384_ntru_plus_kem864	time	0.04997	0.04953	0.00203	0.04975	0.05019	2097
secp521r1	time	0.03766	0.03743	0.00165	0.03754	0.03778	3160
p521_mlkem1024	time	0.04946	0.04909	0.00174	0.04932	0.04960	2108
p521_smaug_t5	time	0.04959	0.04909	0.00274	0.04936	0.04982	2107
p521_ntru_plus_kem1152	time	0.04938	0.04902	0.00237	0.04916	0.04960	2098

Listing A1. Simplified TLS 1.3 handshake benchmark loop (excerpt).

```
#!/usr/bin/env bash
set -euo pipefail

HOST=localhost
PORT=8443
GROUP=p521_ntru_plus_kem1152
RUNS=200
TIME_WINDOW=120

# Fixed-count mode
for i in $(seq 1 $RUNS); do
  t0=$(date +%s.%N)
  echo "Q" | openssl s_client -connect ${HOST}:${PORT} \
      -tls1_3 -groups ${GROUP} \
      -provider default -provider oqsprovider \
      -provider-path "$HOME/oqs-provider_KpqC/build/lib" \
      >/dev/null 2>&1
  t1=$(date +%s.%N)
  echo "$i,$(echo "$t1 - $t0" | bc -l)"
done

# Time-window mode (example: 120 s)
end=$(( $(date +%s) + TIME_WINDOW ))
while [ $(date +%s) -lt $end ]; do
  echo "Q" | openssl s_client -connect ${HOST}:${PORT} \
      -tls1_3 -groups ${GROUP} \
      -provider default -provider oqsprovider \
      -provider-path "$HOME/oqs-provider_KpqC/build/lib" \
      >/dev/null 2>&1
done
```

Note: For full automation (CSV export, warm-up filtering, statistics), see our open repository (https://github.com/minjoo97/oqs-provider_KpqC) (accessed on 16 September 2025).

Appendix A.2. Certificate Generation and Measurement

Listing A2. Excerpt of the certificate generation script (full scripts available in our GitHub repository).

```
#!/usr/bin/env bash
set -euo pipefail

PROV_PATH="$HOME/oqs-provider_KpqC/build/lib"
export OPENSSL_MODULES=$PROV_PATH

ALGOS="prime256v1 secp384r1 secp521r1 haetae2 aimer128s ..."

for algo in $ALGOS; do
  key="${algo}.key"
  crt="${algo}.pem"
  # generate key
  openssl genpkey -provider default -provider oqsprovider \
      -provider-path "$PROV_PATH" -algorithm $algo -out $key
  # self-signed certificate
```

```
  openssl req -new -x509 -key $key -out $crt \
      -subj "/CN=$algo" -days 365 -nodes
done
```

Listing A3. DER-encoded certificate size measurement (excerpt).

```
#!/usr/bin/env bash
set -euo pipefail

ALGOS="prime256v1 secp384r1 secp521r1 haetae2 aimer128s ..."

echo "Algorithm,DER_Bytes"
for algo in $ALGOS; do
  cert="${algo}.pem"
  if [ -f "$cert" ]; then
    size=$(openssl x509 -in "$cert" -outform DER | wc -c | tr -d ' ')
    echo "${algo},${size}"
  fi
done
```

Note: Full automation scripts (covering all supported ECC, NIST PQC, and KpqC algorithms) are provided in our open repository for complete reproducibility.

References

1. Dömösi, P.; Hannusch, C.; Horváth, G. A cryptographic system based on a new class of binary error-correcting codes. *Tatra Mt. Math. Publ.* **2019**, *73*, 83–96. [CrossRef]
2. NIST PQC Project. 2025. Available online: https://csrc.nist.gov/Projects/post-quantum-cryptography (accessed on 7 June 2025).
3. Shor, P.W. Algorithms for quantum computation: Discrete logarithms and factoring. In Proceedings of the Proceedings 35th Annual Symposium on Foundations of Computer Science, Santa Fe, NM, USA, 20–22 November 1994; IEEE: Piscataway, NJ, USA, 1994; pp. 124–134.
4. Grover, L.K. A fast quantum mechanical algorithm for database search. In Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing, Philadelphia, PA, USA, 22–24 May 1996; pp. 212–219.
5. Avanzi, R.; Bos, J.; Ducas, L.; Kiltz, E.; Lepoint, T.; Lyubashevsky, V.; Schanck, J.M.; Schwabe, P.; Seiler, G.; Stehlé, D. CRYSTALS-Kyber algorithm specifications and supporting documentation. *NIST PQC Round* **2019**, *2*, 1–43.
6. Ducas, L.; Kiltz, E.; Lepoint, T.; Lyubashevsky, V.; Schwabe, P.; Seiler, G.; Stehlé, D. Crystals-Dilithium: A lattice-based digital signature scheme. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* **2018**, *2018*, 238–268. [CrossRef]
7. Fouque, P.A.; Hoffstein, J.; Kirchner, P.; Lyubashevsky, V.; Pornin, T.; Prest, T.; Ricosset, T.; Seiler, G.; Whyte, W.; Zhang, Z. *Falcon: Fast-Fourier Lattice-Based Compact Signatures over NTRU*; Submission to the NIST's Post-Quantum Cryptography Standardization Process; 2018; Volume 36. Available online: https://www.di.ens.fr/~prest/Publications/falcon.pdf (accessed on 16 September 2025).
8. Bernstein, D.J.; Hülsing, A.; Kölbl, S.; Niederhagen, R.; Rijneveld, J.; Schwabe, P. The SPHINCS+ signature framework. In Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, London, UK, 11–15 November 2019; pp. 2129–2146.
9. Melchor, C.A.; Aragon, N.; Bettaieb, S.; Bidoux, L.; Blazy, O.; Deneuville, J.C.; Gaborit, P.; Persichetti, E.; Zémor, G.; Bourges, I. Hamming quasi-cyclic (HQC). *NIST PQC Round* **2018**, *2*, 13.
10. NIST PQC Project: Digital Signature Schemes. 2025. Available online: https://csrc.nist.gov/Projects/pqc-dig-sig/round-1-additional-signatures (accessed on 7 June 2025).
11. KpqC Competition. 2025. Available online: https://kpqc.or.kr/competition.html (accessed on 7 June 2025).
12. Cheon, J.H.; Choe, H.; Devevey, J.; Güneysu, T.; Hong, D.; Krausz, M.; Land, G.; Möller, M.; Stehlé, D.; Yi, M. HAETAE: Shorter Lattice-Based Fiat-Shamir Signatures. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* **2024**, *2024*, 25–75. [CrossRef]
13. Kim, S.; Ha, J.; Lee, J. AIM: Symmetric Primitive for Shorter Signatures with Stronger Security. In Proceedings of the ACM CCS 2023: ACM Conference on Computer and Communications Security, Copenhagen, Denmark, 26–30 November 2023; ACM (Association for Computing Machinery): New York, NY, USA, 2023.
14. SMAUG-T: The Key Exchange Algorithm Based on Module-LWE and Module-LWR. 2025. Available online: https://kpqc.or.kr/images/pdf/SMAUG-T_Document.pdf (accessed on 7 June 2025).

15. Kim, J.; Park, J.H. NTRU+: Compact construction of NTRU using simple encoding method. *IEEE Trans. Inf. Forensics Secur.* **2023**, *18*, 4760–4774. [CrossRef]
16. Chen, A.C. Post-Quantum Cryptography X. 509 Certificate. In Proceedings of the 2024 International Conference on Smart Systems for Applications in Electrical Sciences (ICSSES), Tumakuru, India, 3–4 May 2024; IEEE: Piscataway, NJ, USA, 2024; pp. 1–6.
17. Kampanakis, P.; Panburana, P.; Daw, E.; Van Geest, D. The viability of post-quantum X. 509 certificates. *Cryptol. ePrint Arch.* **2018**. Available online: https://eprint.iacr.org/2018/063.pdf (accessed on 16 September 2025).
18. Garrach, M.A.; Waghela, C.; Mathews, M.M.; Sreekuttan, L.S. Benchmarking Speed of Post-Quantum Lattice Based PKE/KEM Schemes Using Liboqs. In Proceedings of the 2022 International Conference on Trends in Quantum Computing and Emerging Business Technologies (TQCEBT), Pune, India, 13–15 October 2022; IEEE: Piscataway, NJ, USA, 2022; pp. 1–5. [CrossRef]
19. Opiłka, F.; Niemiec, M.; Gagliardi, M.; Kourtis, M.A. Performance Analysis of Post-Quantum Cryptography Algorithms for Digital Signature. *Appl. Sci.* **2024**, *14*, 4994. [CrossRef]
20. Open Quantum Safe. Available online: https://openquantumsafe.org/ (accessed on 7 June 2025).
21. liboqs. Available online: https://github.com/open-quantum-safe/liboqs (accessed on 7 June 2025).
22. Housley, R.; Ford, W.; Polk, W.; Solo, D. RFC 5280: Internet x. 509 Public Key Infrastructure Certificate and CRL Profile. 2008. Available online: https://datatracker.ietf.org/doc/html/rfc5280 (accessed on 16 September 2025).
23. Migration to Post-Quantum Cryptography Quantum Readiness: Testing Draft Standards. 2023. Available online: https://www.nccoe.nist.gov/sites/default/files/2023-12/pqc-migration-nist-sp-1800-38c-preliminary-draft.pdf (accessed on 7 June 2025).
24. pq-hybrid-x509. 2025. Available online: https://datatracker.ietf.org/doc/draft-truskovsky-lamps-pq-hybrid-x509/ (accessed on 7 June 2025).
25. IETF Chameleon Certificates. 2025. Available online: https://datatracker.ietf.org/doc/draft-bonnell-lamps-chameleon-certs/ (accessed on 7 June 2025).
26. IETF Composite Certificates. 2025. Available online: https://datatracker.ietf.org/doc/draft-ounsworth-pq-composite-sigs/09/ (accessed on 7 June 2025).
27. Sim, M.; Eum, S.; Song, G.; Lee, M.; Kim, S.; Song, M.; Seo, H. KpqClean Ver2: Comprehensive Benchmarking and Analysis of KpqC Algorithm Round 2 Submissions. *Cryptol. ePrint Arch.* **2024**. Available online: https://eprint.iacr.org/2024/1288 (accessed on 16 September 2025).
28. Kwon, H.; Sim, M.; Song, G.; Lee, M.; Seo, H. Evaluating kpqc algorithm submissions: Balanced and clean benchmarking approach. In Proceedings of the International Conference on Information Security Applications, Jeju Island, Republic of Korea, 23–25 August 2023; Springer: Singapore, 2023; pp. 338–348.
29. Choi, Y.; Kim, M.; Kim, Y.; Song, J.; Jin, J.; Kim, H.; Seo, S.C. KpqBench: Performance and Implementation Security Analysis of KpqC Competition Round 1 Candidates. *IEEE Access* **2024**, *12*, 18606–18626. [CrossRef]