An Investigation into Post-Quantum KEM Deployment Suitability

Rama Ayman Alshar'
Virginia Tech
Blacksburg, Virginia
alsharerama@vt.edu

William Furgerson
Virginia Tech
Alexandria, Virginia
williamfurgerson@vt.edu

Conor McFadden
Virginia Tech
Alexandria, Virginia
conormcfadden@vt.edu

Mihir Umeshchandra Patel
Virginia Tech
Alexandria, Virginia
mihirpatel@vt.edu

***Abstract*—The transition to post-quantum cryptography requires replacing classical elliptic-curve key exchange in TLS with quantum-resistant alternatives. This paper evaluates the two NIST-selected Key Encapsulation Mechanisms (KEMs), CRYSTALS-Kyber and HQC, to determine their suitability as drop-in replacements for ECDHE in HTTPS. We define practical deployment requirements based on TLS handshake constraints, including message size limits derived from the standard MTU, constant-time behavior, and compatibility with common hardware platforms. Using these requirements, we analyze each algorithm's structural properties and benchmark their performance across ARM, desktop-class x86, and server-class x86 CPUs.**

***Index Terms*—component, formatting, style, styling, insert**

I. Introduction

Today, Internet security relies heavily on HTTPS, which in turn depends on the SSL and TLS protocols to protect end-to-end communication. Within TLS, Key Encapsulation Mechanisms (KEMs) are used to establish shared secret keys over untrusted networks. Once a shared key is established, all further communication is protected using symmetric cryptography. In TLS 1.3, this key agreement is primarily performed using Elliptic Curve Diffie–Hellman Ephemeral (ECDHE). ECDHE is efficient and widely deployed, but it is vulnerable to attack by a sufficiently powerful quantum computer.

Although large-scale quantum computers do not yet exist, progress in quantum computing is rapid and poses a realistic long-term risk to current public-key cryptography. Shor's algorithm allows a quantum computer to break RSA by efficiently factoring large integers and can also break Diffie–Hellman and elliptic curve systems by solving discrete logarithm problems. If classical key exchange algorithms remain in HTTPS when practical quantum machines become available, encrypted web traffic, including online banking and e-commerce, could be decrypted. To address this risk, researchers have developed quantum-resistant alternatives that are not based on mathematical problems vulnerable to known quantum attacks. In response, the National Institute of Standards and Technology (NIST) launched a multi-year standardization process [1] for Post-Quantum Cryptography (PQC). This process resulted in the selection of five algorithms for standardization, including two KEMs and three digital signature schemes.

As these standards mature, system designers and operators must determine whether the new algorithms are practical replacements for classical mechanisms in real-world deployments. Prior work has focused mainly on the performance and security of standardized signature algorithms [2] [3], while the performance and operational impact of post-quantum KEMs in TLS has received less attention. The goal of this paper is to measure the performance of the NIST-selected KEMs on contemporary hardware and to evaluate how well they integrate with existing HTTPS workflows. This work aims to provide concrete data to support deployment decisions, identify practical challenges, and assess which algorithms are most suitable for different application scenarios.

II. Prior Work

Post-quantum cryptography has been studied across a wide range of systems, including cyber-physical systems [6], blockchain platforms [7], embedded systems [8], and the Internet of Things (IoT) [9]. As interest in these applications grew, researchers began measuring the performance cost of integrating post-quantum algorithms into real protocols and platforms.

Several studies proposed frameworks to evaluate the overhead of adding PQC to TLS [10]. These works used OpenSSL to benchmark KEMs and signature schemes and measured both computational cost and message size during the TLS handshake. They reported that large post-quantum signatures introduce challenges compared to traditional TLS configurations. However, their evaluations did not include all NIST-selected KEMs.

Other work examined hybrid post-quantum TLS 1.3 deployments [8]. These studies measured end-to-end latency on constrained embedded devices and showed that large post-quantum signatures increase handshake time. They did not evaluate post-quantum KEMs or signatures as standalone replacements for classical mechanisms, and instead focused on combined hybrid constructions.

Additional research evaluated both security and performance [11]. This work integrated the Open Quantum Safe (OQS) project into a modified TLS stack and tested it under real network conditions. The results showed improved resistance to quantum attacks but also reported higher handshake latency in practice. Other studies implemented PQC-enabled TLS using OQS and OpenSSL [12]. By varying network conditions such as latency and packet loss, they showed that

network delay can hide raw computational overhead. They also observed that when packet loss exceeds 3% to 5%, large keys that require fragmentation reduce connection performance, especially in unstructured lattice-based schemes. Their work focused on specific key sizes and did not compare multiple parameter sets within the same algorithm family.

All in all, prior research has focused mainly on hybrid TLS configurations that combine classical and post-quantum algorithms. Many studies evaluated only a subset of the NIST-selected candidates, and several considered only a single parameter set. This leaves open questions about how key size, network conditions, and resource usage interact when classical KEMs are fully replaced. In this work, we study standalone post-quantum KEMs where the classical key exchange is entirely replaced. We evaluate all NIST-selected KEMs across all standardized parameter sets and measure performance differences across algorithms, parameter choices, and hardware platforms.

III. TLS KEM Requirements

Before evaluating performance, it is necessary to define the practical requirements a post-quantum KEM must satisfy to function as a drop-in replacement for ECDHE in TLS.

The first and most important constraint is message size. TLS handshakes are carried over TCP/IP, and most Internet paths are optimized around a standard Ethernet Maximum Transmission Unit (MTU) of 1500 bytes. When a single TLS handshake message exceeds this limit, the packet is fragmented or retransmitted, which increases latency and reduces reliability. To avoid fragmentation in typical deployments, the KEM public key and ciphertext must fit comfortably within a single MTU along with the rest of the TLS handshake fields.

A TLS 1.3 ClientHello and ServerHello already include protocol versioning, extensions, cipher suite negotiation, and key share structures. In practice, these fields commonly consume several hundred bytes. To leave sufficient room for headers and extensions, we define a conservative size budget of approximately 1,200 bytes for any single KEM payload carried in the handshake. This threshold ensures that the KEM public key and ciphertext can be transmitted without fragmentation on standard networks, while still allowing space for the surrounding TLS structures.

The second requirement is platform compatibility. TLS is used across virtually all modern computing platforms, including x86 servers, ARM-based mobile devices, embedded systems, and network appliances. A viable KEM must therefore have reference and optimized implementations that run correctly on both x86 and ARM architectures, without relying on exotic hardware features. A mechanism that only performs acceptably on a single CPU family would not be suitable for broad TLS deployment.

The third requirement is resistance to timing side channels. Since TLS handshakes occur in adversarial network environments, implementations must not leak secret information through timing behavior. Constant-time execution is therefore a strict requirement. Both KEMs selected through the NIST post-quantum process satisfy this property by design and through standardization requirements [1]. This allows us to treat timing safety as a baseline property rather than a variable in our evaluation.

These constraints define the non-negotiable properties required for post-quantum KEMs in real TLS deployments: message sizes that fit safely under the MTU, portability across common hardware platforms, and constant-time behavior.

IV. NIST-Selected KEMs

This work focuses on the two Key Encapsulation Mechanisms (KEMs) selected by NIST for standardization: CRYSTALS-Kyber and HQC [1]. Both algorithms are designed to resist classical and quantum attacks and are intended as drop-in replacements for classical key exchange mechanisms such as ECDHE in TLS.

TABLE I
KEM data sizes (bytes)

Level I	Public Key	Private Key	Ciphertext
Kyber512	800	1632	768
ECDH-256	65	32	65
HQC-128	2249	2289	4497
Level III	Public Key	Secret Key	Ciphertext
Kyber768	1184	2400	1088
HQC-192	4522	4562	9042
ECDH-384	97	48	97
Level V	Public Key	Secret Key	Ciphertext
Kyber1024	1568	3168	1568
HQC-256	7245	7285	14485
ECDH-521	133	66	133

A. CRYSTALS-Kyber

Kyber is a lattice-based KEM whose security is based on the hardness of problems related to structured lattices, in particular the Learning With Errors (LWE) problem [4]. In Kyber, the receiver generates a public and private key pair using lattice-based equations. The sender uses the receiver's public key to construct a ciphertext that encapsulates a freshly generated shared secret. During this process, the secret is combined with a small, carefully structured random error. An attacker attempting to recover the shared secret must solve the underlying LWE problem or recover the injected noise, which is considered infeasible even for quantum adversaries. The legitimate receiver, however, can efficiently remove the noise and recover the shared secret using the private key.

From a deployment perspective, Kyber's message sizes are significantly larger than classical ECDH, but remain within a range that can fit inside a single MTU when combined with the rest of the TLS handshake. As shown in Table I, Kyber512 produces an 800-byte public key and a 768-byte ciphertext. Kyber768 produces a 1184-byte public key and a 1088-byte ciphertext. Both of these fit within the 1,200-byte size budget defined by the MTU-derived TLS constraint. At the highest parameter set, Kyber1024 produces a 1568-byte public key

and a 1568-byte ciphertext, which exceed this budget and are more likely to cause fragmentation during TLS handshakes.

B. Hamming Quasi-Cyclic (HQC)

HQC is a code-based KEM built on the hardness of decoding structured linear error-correcting codes, specifically the Quasi-Cyclic Syndrome Decoding (QCSD) problem [5]. In HQC, the receiver generates a key pair in which the private key consists of sparse secret vectors with low Hamming weight, and the public key is derived from these secrets using structured generator matrices. To encapsulate a key, the sender combines the receiver's public key with randomly chosen vectors, an injected error, and an encoded message to form the ciphertext. The legitimate receiver uses error-correction techniques to recover the encoded message and derive the shared secret. An attacker, by contrast, would need to recover the sparse error structure, which is computationally infeasible under the QCSD assumption.

The data sizes of HQC are significantly larger than both classical and lattice-based alternatives. As shown in Table I, HQC-128 uses a 2249-byte public key and a 4497-byte ciphertext. HQC-192 increases to a 4522-byte public key and a 9042-byte ciphertext. HQC-256 further expands to a 7245-byte public key and a 14485-byte ciphertext. All of these values greatly exceed the 1,200-byte target derived from the TLS MTU constraint and would require fragmentation in standard TLS handshakes.

While HQC does not meet the size goals defined in the requirements, size is the least strict constraint if HQC is the only candidate that satisfies all other functional and security requirements. Whether this trade-off is acceptable will be determined after the performance and integration evaluations in later sections.

V. Synthetic Benchmarking

A. Methodology

To evaluate the raw cryptographic performance of each KEM and parameter set, we built a dedicated benchmarking suite that repeatedly executes key generation, encapsulation, and decapsulation operations. The design of the suite follows the general methodology of the System for Unified Performance Evaluation Related to Cryptographic Operations and Primitives (SUPERCOP) [14] and aligns with prior work on post-quantum TLS benchmarking by Paquin et al. [12] and Sikeridis et al. [15].

We tested across three hardware platforms to represent the main classes of systems that terminate TLS connections in real deployments. First, Apple M1 was selected as a representative ARM-based system. This class reflects modern mobile devices, tablets, and newer laptops. AMD Ryzen 9 5900X was selected to represent a desktop-class x86 CPU, and an Intel Xeon Gold 5218 was selected to represent a server-class CPU, typical of data center TLS termination. These platforms allow us to evaluate how well each algorithm scales across the types of hardware that dominate TLS usage in practice.

Each benchmark run begins with a warm-up phase. This stabilizes processor frequency under dynamic voltage and frequency scaling and primes instruction and data caches so that measurements reflect steady-state behavior rather than cold-start effects. Similar warm-up techniques are standard in cryptographic benchmarking [16] [12] [17].

For each algorithm and parameter set, we performed 10,000 timed iterations of each operation. This sample size provides enough statistical power to characterize both average behavior and spread. Prior work such as SUPERCOP [14], Paquin et al. [12], and Sikeridis et al. [15] uses similar iteration counts to reduce measurement noise. Repeating operations at this scale reduces the impact of transient system events such as OS scheduling, interrupts, and cache evictions.

After data collection, we removed extreme outliers and computed medians, percentiles, and standard deviation. This approach captures realistic steady-state performance rather than best-case or worst-case behavior.

B. Implementation

Following this methodology, we implemented a standalone benchmarking suite. The full source code is available in our repository on GitHub [18]. The suite integrates the NIST reference implementations of Kyber and HQC [1] and wraps OpenSSL's ECDH implementation to conform to the NIST KEM API for a baseline comparison.

Kyber was tested in both its reference implementation and its AVX-optimized variant. This allows us to see the performance difference between machines with AVX instructions and machines without. All algorithms were compiled and tested at NIST security levels I, III, and V. All implementations expose identical function signatures through NIST's standardized api.h interface, allowing the same harness to be used without algorithm-specific logic.

The harness measures key generation, encapsulation, and decapsulation. Warm-up iterations are performed before measurement iterations, and the tool automatically discards statistical outliers before computing final metrics.

C. Results

After running the benchmarks, we observed that across all algorithms the spread of runtimes are very tight. As expected from constant time execution being a NIST requirement, there is almost no difference from 25th percentile to 75th percentile. Because of this, we display in this report only the median values. For those interested in a complete distribution, our testing suite is available on github [18] and reports full metrics. Table II shows the measured throughput in KiloHandshakes per second for each platform and security level.

ECDH performed consistently across platforms but at significantly lower throughput than Kyber for all security levels. As our baseline, currently in deployment KEM, ECDHs performance will be used as our "definition of normal" for Kyber and ACDH to be compared against.

Across all three security levels, Kyber achieved the highest throughput on all platforms. The AVX-optimized Kyber

TABLE II
KEM KILOHANDSHAKES PER SECOND

Level I	Apple M1	Ryzen 9 5900X	Xeon Gold 5218
Kyber512 AVX	N/A	58.4	34.8
Kyber512	41.6	24.1	12.2
ECDH-256	N/A	9.8	4.5
HQC-128	9.9	11.5	6.4
Level III	Apple M1	Ryzen 9 5900X	Xeon Gold 5218
Kyber768 AVX	N/A	48.9	23.3
Kyber768	26.3	13.9	8.6
HQC-192	N/A	4.6	2.1
ECDH-384	1.8	1.3	0.8
Level V	Apple M1	Ryzen 9 5900X	Xeon Gold 5218
Kyber1024 AVX	N/A	27.8	17.6
Kyber1024	16.3	9.8	5.7
HQC-256	N/A	2.3	1.0
ECDH-521	1.8	1.7	0.9

variants significantly increased throughput on x86 systems, roughly doubling performance compared to the baseline implementations. This reflects Kyber's ability to efficiently use vectorized polynomial arithmetic. The Apple M1 results show that the baseline Kyber implementations run efficiently on ARM, achieving competitive performance relative to x86 systems. In contrast, the AVX-optimized Kyber variants are not available on ARM, as expected, because AVX is specific to x86 processors. However, kyber still meets the platform portability requirement defined earlier, since a reference implementation exists and performs well on non-x86 hardware.

HQC showed worse hardware compatibility. We were unable to run the HQC implementation on the Apple M1 platform in our environment. This lack of operational ARM support represents a practical limitation relative to the earlier requirement that a TLS-suitable KEM must be deployable across common hardware platforms. Additionally, even on the x86 systems where it did run, HQC showed lower throughput than Kyber across all security levels. This is not to say that HQC performed badly (it performed very comparably to ECDH), Kyber just set a very high bar by exceeding expectations.

Overall, the results show that Kyber provides strong throughput across desktop, server, and ARM-based systems using its portable implementation, while also offering substantial gains on x86 platforms through vectorized code. HQC demonstrates functional performance on x86 systems but shows gaps in practical cross-platform support under our test conditions.

VI. END-TO-END BENCHMARKING

A. Methodology

While synthetic benchmarking is useful for comparing raw KEM behavior, it does not show how post-quantum key exchange affects the full TLS handshake. In practice, TLS handshake latency is dominated by certificate verification, key schedule derivation, memory management, and other protocol operations. Measuring the complete handshake therefore allows us to evaluate whether post-quantum KEMs introduce any user-visible delays when integrated into existing HTTPS workflows.

Based on the synthetic results presented earlier, Kyber was selected for end-to-end testing because it was the only NIST-standardized KEM that met the size requirement, ran correctly on all tested hardware, and consistently outperformed both HQC and classical ECDH in computational benchmarks. HQC, while secure, does not meet the MTU-based size requirement and lacked working ARM support in our environment, making it unsuitable for practical TLS evaluation.

To study real-world behavior, we performed two sets of TLS 1.3 handshake measurements using OpenSSL. The first used OpenSSL's default ECDHE key exchange, providing a baseline that reflects current web deployments. The second used the oqs-provider [19] to substitute Kyber768 for key exchange under identical system conditions.

After a short warm-up, each configuration executed 30 consecutive TLS handshakes using openssl s_client. For each connection, we recorded wall-clock time from initiation to handshake completion. Running multiple trials allowed us to smooth out OS scheduling noise and other transient effects. All tests were performed on localhost to isolate TLS-internal costs from network latency.

B. Results

The results show that Kyber did not introduce any additional latency into the TLS handshake. Instead, Kyber produced a consistently faster median handshake time. Across the 30-run series, the ECDHE configuration measured a median of 73 ms, while the Kyber-based handshake completed in 55 ms.

This behavior aligns with the synthetic results in Table III, which show encapsulation and decapsulation times in the tens of microseconds. These microsecond-scale costs are negligible relative to the tens of milliseconds spent inside a full TLS handshake. As a result, differences in key exchange computation have almost no influence on the final user-visible latency.

TABLE III
KEM SYNTHETIC TIME (μS)

Security Level I	Apple M1	Ryzen 9 5900X	Xeon Gold 5218
Kyber512 AVX	N/A	15	29
Kyber512	24	41	82
HQC-128	N/A	99	224
ECDH-256	100	96	156
Security Level III	Apple M1	Ryzen 9 5900X	Xeon Gold 5218
Kyber768 AVX	N/A	27	43
Kyber768	38	71	116
HQC-192	N/A	215	486
ECDH-384	543	761	1264
Security Level V	Apple M1	Ryzen 9 5900X	Xeon Gold 5218
Kyber1024 AVX	N/A	32	57
Kyber1024	60	90	172
HQC-256	N/A	437	978
ECDH-521	559	660	1190

These findings confirm that the cryptographic component of the handshake—whether classical or post-quantum—makes up

only a tiny fraction of total latency. Certificate processing and internal library overhead dominate both measurements, and the slight advantage seen with Kyber likely arises from differences in internal OpenSSL code paths rather than the KEM itself.

In real deployments, where network round-trip times will further increase the cost of any handshake, the relative impact of the KEM shrinks even further. Overall, the end-to-end experiments show that adopting PQC in TLS key exchange does not impair connection latency and can, under some conditions, outperform the classical mechanisms it is intended to replace.

VII. Recommendations

Overall, our evaluation shows that post-quantum key exchange can be integrated into TLS today, but only one of the standardized algorithms meets the practical requirements defined earlier in this paper.

Kyber at security levels I and III meets all of the requirements. Both parameter sets keep their public key and ciphertext sizes within the MTU-derived limits and therefore avoid fragmentation during the handshake. They also deliver high throughput and perform reliably across platform architectures. These properties make Kyber a practical replacement for ECDHE in real-world TLS deployments. The AVX-optimized variant further improves throughput on x86 systems without affecting portability.

Kyber at security level V does not fully meet the size requirement, since its public key and ciphertext exceed the safe budget for a single unfragmented TLS message. However, its performance remains strong, and its sizes are substantially smaller than other level V options. For deployments that require maximum post-quantum security and can tolerate occasional fragmentation, Kyber1024 is the only viable candidate among the standardized KEMs.

HQC does not meet the requirements for TLS integration. Its message sizes exceed the MTU budget at all security levels, causing unavoidable fragmentation and additional handshake latency. In our testing environment, HQC also lacked functional support on ARM-based hardware, which limits its deployability across the diverse systems that use TLS today. While HQC satisfies the cryptographic security and constant-time requirements set by NIST, these operational limitations make it unsuitable for widespread TLS adoption in its current form.

Finally, it seems that vectorized instructions will remain important for post-quantum performance. Kyber's AVX implementation showed a clear improvement on x86 CPUs, and lattice-based designs are likely to continue benefiting from vector-friendly architectures. Future hardware planning should therefore favor systems with robust SIMD support, as this will improve performance for both Kyber and potential successor algorithms.

References

[1] Computer Security Division, Information Technology Laboratory, National Institute of Standards and Technology, U.S. Department of Commerce, "Selected Algorithms - Post-Quantum Cryptography — CSRC — CSRC." https://csrc.nist.gov/Projects/post-quantum-cryptography/pqc-standardization/selected-algorithms

[2] M. Müller, J. De Jong, M. Van Heesch, B. Overeinder, and R. Van Rijswijk-Deij, "Retrofitting post-quantum cryptography in internet protocols," ACM SIGCOMM Computer Communication Review, vol. 50, no. 4, pp. 49–57, Oct. 2020, doi: 10.1145/3431832.3431838.

[3] M. Raavi, Q. Khan, S. Wuthier, P. Chandramouli, Y. Balytskyi, and S.-Y. Chang, "Security and performance analyses of Post-Quantum digital signature algorithms and their TLS and PKI integrations," Cryptography, vol. 9, no. 2, p. 38, Jun. 2025, doi: 10.3390/cryptography9020038.

[4] O. Regev, "Lattice-Based cryptography," in Lecture notes in computer science, 2006, pp. 131–141. doi: 10.1007/11818175_8.

[5] F. Antognazza, A. Barenghi, and G. Pelosi, "An efficient and unified RTL accelerator design for HQC-128, HQC-192, and HQC-256," IEEE Transactions on Computers, vol. 74, no. 7, pp. 2306–2320, Apr. 2025, doi: 10.1109/tc.2025.3558044.

[6] S. Paul and P. Scheible, "Towards post-quantum security for cyber-physical systems: Integrating PQC into industrial M2M communication," Journal of Computer Security, vol. 30, no. 4, pp. 623–653, Nov. 2021, doi: 10.3233/jcs-210037.

[7] M. Buser et al., "A survey on Exotic signatures for Post-quantum Blockchain: Challenges and research directions," ACM Computing Surveys, vol. 55, no. 12, pp. 1–32, Dec. 2022, doi: 10.1145/3572771.

[8] D. Marchsreiter and J. Sepulveda, "Hybrid Post-Quantum Enhanced TLS 1.3 on embedded devices," 2022 25th Euromicro Conference on Digital System Design (DSD), pp. 905–912, Aug. 2022, doi: 10.1109/dsd57027.2022.00127.

[9] M. Schöffel, F. Lauer, C. C. Rheinländer, and N. Wehn, "Secure IoT in the era of Quantum Computers—Where are the bottlenecks?," Sensors, vol. 22, no. 7, p. 2484, Mar. 2022, doi: 10.3390/s22072484.

[10] J. A. Montenegro, R. Rios, and J. López-Cerezo, "A performance evaluation framework for post-quantum TLS," Future Generation Computer Systems, vol. 175, p. 108062, Jul. 2025, doi: 10.1016/j.future.2025.108062.

[11] D. Stebila and M. Mosca, "Post-quantum Key Exchange for the Internet and the Open Quantum Safe Project," in Lecture notes in computer science, 2017, pp. 14–37. doi: 10.1007/978-3-319-69453-5_2.

[12] J. Ding and J.-P. Tillich, Post-Quantum cryptography: 11th International Conference, PQCrypto 2020, Paris, France, April 15–17, 2020, Proceedings. Springer Nature, 2020.

[13] "RFC 8446: The Transport Layer Security (TLS) Protocol Version 1.3," IETF Datatracker. https://datatracker.ietf.org/doc/rfc8446/

[14] "SUPERCOP." https://bench.cr.yp.to/supercop.html

[15] Dimitrios Sikeridis, Panos Kampanakis, and Michael Devetsikiotis. Post-quantum authentication in TLS 1.3: A performance study. In Network and Distributed Systems Security (NDSS) Symposium 2020, San Diego, CA, USA, February 2020. The Internet Society. doi: 10.14722/ndss.2020.24203.

[16] Andy and Andy, "Fundamentals of Software Optimization Part I — Benchmarking - data, code and conversation," data, code and conversation - from Andy Boothe, Apr. 29, 2022. https://sigpwned.com/2022/04/16/fundamentals-of-software-optimization-part-i-benchmarking/

[17] Dimitrios Sikeridis, Panos Kampanakis, and Michael Devetsikiotis. Assessing the overhead of post-quantum cryptography in TLS 1.3 and SSH. In Proceedings of the 16th International Conference on emerging Networking EXperiments and Technologies (CoNEXT '20), pages 149–156, Barcelona, Spain, December 2020. ACM. doi: 10.1145/3386367.3431305.

[18] MovementGH, "GitHub - MovementGH/pqc-kem-benchmarking," GitHub. https://github.com/MovementGH/pqc-kem-benchmarking

[19] Open-Quantum-Safe, "GitHub - open-quantum-safe/oqs-provider: OpenSSL 3 provider containing post-quantum algorithms," GitHub. https://github.com/open-quantum-safe/oqs-provider

Appendix A
Contributions

Below are the responsibilities and contributions of each team member to the project:

Conor McFadden:

- Synthetic benchmarking methodology
- Synthetic benchmarking test harness

Mihir Patel:

- End to end testing methodology
- End to end testing implementation

Rama Alshar':

- Literature review
- Algorithm research
- Requirements analysis

William Furgerson:

- PQC algorithm build system and harness integration
- ECDHE reference implementation
- Synthetic Data collection
- Paper and presentation editing / formatting,