Simulation-based Evaluation of Post-Quantum Algorithm in TLS 1.3

Hyun-jun Kim[1], Si-woo Eum[2], Min-ho Song[3], and Hwa-jeong Seo[4†]

[1] Hansung University, Seoul, South Korea
khj930704@gmail.com
[2] Hansung University, Seoul, South Korea
shuraatum@gmail.com
[3] Hansung University, Seoul, South Korea
smino0906@gmail.com
[4] Hansung University, Seoul, South Korea
hwajeong84@gmail.com

Abstract

The rapid advancement of quantum computing poses a significant threat to traditional cryptographic methods, highlighting the urgent need for post-quantum cryptographic solutions. This study evaluates the efficacy of TLS 1.3, which incorporates post-quantum cryptography, using the NS-3 simulation tool. Simulations under diverse network conditions, including variations in latency and bandwidth, allow us to assess the impact of post-quantum cryptography on each phase of TLS 1.3. The research also simulates both point-to-point and Wi-Fi network environments, offering key insights into TLS 1.3's behavior across different network setups. By delineating the integration of quantum-resistant cryptography in TLS 1.3, this study deepens our understanding of its influence on system performance and assists in the selection of the most suitable encryption algorithms.

1 Introduction

The emergence of quantum computing, with its capability to efficiently break traditional cryptographic methods, presents a rising threat to conventional cipher systems [16, 13]. Consequently, there's an urgent need to transition to quantum-resistant cryptographic algorithms. However, such a transition is intricate and time-consuming. As outlined by NIST, this process encompasses modifications in cryptographic libraries, validation tools, and hardware. It underscores the necessity to thoughtfully consider implementation, key size, and performance during this shift [3].

Incorporating quantum-resistant cryptography into prevalent security protocols demands not only an understanding of its integration but also an investigation into its subsequent impact on system performance. A prime focus of ongoing research is the integration of the Transport Layer Security (TLS) protocol — a cornerstone for ensuring data privacy and integrity between two communicating entities — with Post-Quantum Cryptography (PQC). Current studies shed light on the security and performance attributes of several cryptographic systems, the intricacies of their amalgamation, and how varying network conditions influence their operation.

Nevertheless, despite the substantial insights provided by these studies, certain knowledge gaps persist. Many extant analyses lean heavily towards real-world evaluations or specific embedded platforms. The potential insights from a wholly controlled, simulated environment — where variables inherent to real-world networks can be individually manipulated — remain largely uncharted.

To bridge this void and further the groundwork laid by antecedent research, this study offers an exhaustive performance analysis of TLS 1.3 integrated with quantum-resistant cryptography,

harnessing the capabilities of the NS-3 simulation tool. Within this controlled simulation, we can meticulously dissect the impacts of various PQC encryption methods on TLS 1.3 across different network scenarios. By adjusting elements like network latency, packet loss, and bandwidth constraints, the simulation furnishes a deeper comprehension of their influence on performance. Our contributions include an in-depth performance assessment of TLS 1.3 utilizing quantum-resistant encryption in a simulated environment. Moreover, we conducted simulations in both point-to-point and Wi-Fi network scenarios. These simulations aimed to elucidate the efficacy of various Post-Quantum Cryptography (PQC) encryption algorithms, with each showcasing its unique strengths and limitations.

2 Related Works

2.1 NIST PQC Standardization Process Algorithms

NIST(National Institute of Standards and Technology) conducted the NIST PQC (Post-Quantum Cryptography) competition to safeguard encryption technology from the advent of quantum computers. [1] Through 3 rounds, the algorithms for PQC standardization in 2022 were selected, and Round 4 is currently underway to further enhance the diversity of the adopted post-quantum cryptographic foundations. Kyber [6] was selected as the key exchange algorithm, owing to its efficient computation speed and compact key size. For the signature operation, three algorithms Dilithium [10], Falcon [11], and SPHINCS+ [5] were chosen. Each of these algorithms has its unique advantages and challenges. Dilithium excels with fast signature and verification speed, which is crucial for real-time systems. Falcon, on the other hand, is appreciated for its high speed and small key size, making it particularly suitable for high-performance cryptographic systems. SPHINCS+, a stateless hash-based algorithm, offers a higher degree of security than its counterparts, which is critical when the highest data integrity is demanded. Currently, Classic McEliece [4], BIKE [2], HQC [14], and SIKE [12] are undergoing the additional 4th round as candidates. Among them, it has been revealed that SIKE has a security vulnerability and is potentially being excluded [8]. Classic McEliece, albeit offering robust security, grapples with the limitation of a large key size, which could affect storage and transmission efficiency. BIKE, leveraging error correction codes, presents high compatibility with existing systems, an attribute that facilitates smoother transition and integration. HQC offers a relatively higher security level, a trade-off for potentially slower computation speeds. These algorithms, each with varying key sizes, significantly influence factors such as network performance, storage space, and computation speed. Therefore, the selection process must carefully balance these considerations, given the specific needs and resources of the systems in question. Table 1 is a parameter set summary of Key encapsulation mechanisms (KEMs) and Table 2 is a parameter set summary of Signature schemes.

2.2 TLS 1.3

The Transport Layer Security (TLS) protocol is a cryptographic protocol designed to provide secure communication over a network. Given its wide adoption across various applications - web browsing, email, instant messaging, to name just a few - TLS is a critically important protocol for securing digital communications. TLS 1.3 consists of several important steps in the handshake process, such as ClientHello, ServerHello, key exchange, server and client authentication, and Finished messages. Each of these stages plays a vital role in ensuring the communication security between two parties. The handshake begins with the ClientHello message, wherein the

Parameter set	Security model	Claimed NIST Level	Public key size (bytes)	Secret key size (bytes)	Ciphertext size (bytes)	Shared secret size (bytes)
Kyber512	IND-CCA2	1	800	1632	768	32
Kyber768	IND-CCA2	3	1184	2400	1088	32
Kyber1024	IND-CCA2	5	1568	3168	1568	32
BIKE-L1	IND-CPA	1	1541	5223	1573	32
BIKE-L3	IND-CPA	3	3083	10105	3115	32
BIKE-L5	IND-CPA	5	5122	16494	5154	32
HQC-128	IND-CCA2	1	2249	2289	4481	64
HQC-192	IND-CCA2	3	4522	4562	9026	64
HQC-256	IND-CCA2	5	7245	7285	14469	64

Table 1: Parameter set summary of Key encapsulation mechanisms

client communicates to the server the necessary parameters for encryption and the available encryption algorithms. Following this, the ServerHello message sees the server selecting an encryption algorithm and providing its response. This stage sets the groundwork for the impending key exchange and secure communication. Key exchange in TLS 1.3 typically uses mechanisms like Diffie-Hellman Ephemeral (DHE) or Elliptic Curve Diffie-Hellman Ephemeral (ECDHE). At this stage, the Hybrid Key Exchange in TLS 1.3 becomes relevant, merging classical and post-quantum algorithms for enhanced protection against potential quantum threats [18]. This approach not only ensures future-proof security but also retains compatibility with existing infrastructure. After solidifying the key exchange, both the client and server conclude the handshake by sending Finished messages. These messages, containing a cryptographic hash of the handshake process, certify the communication's authenticity and integrity.

2.3 Post-Quantum TLS 1.3

Various studies have been conducted to integrate and evaluate post-quantum algorithms into TLS 1.3. Stebila et al. [19] discuss the security and performance characteristics of BCNS15 and Frodo, both individually and in the context of the Transport Layer Security (TLS) protocol. Furthermore, they evaluate the security, performance, and integration of these protocols with the TLS protocol through the Open Quantum Safe project, which provides an open-source platform for experimenting with quantum-resistant cryptography. crockett et al. [9] delves into the challenges of implementing post-quantum and hybrid key exchange and authentication in TLS 1.2, TLS 1.3, and SSH, emphasizing the issue of size limits for key exchange and signatures. Adjusting these size limits could potentially impact performance and resistance to denial-of-service attacks, and while initial results are positive, deployment of these changes might necessitate careful negotiation and pose risks of compatibility issues. Bürstinghaus-Steinbach et al. [7] integrates the post-quantum KEM scheme Kyber and the post-quantum signature scheme SPHINCS+ into the mbed TLS library and measures the performance on four

Parameter set	Security model	Claimed NIST Level	Public key size (bytes)	Secret key size (bytes)	Signature size (bytes)
Dilithium2	EUF-CMA	2	1312	2528	2420
Dilithium3	EUF-CMA	3	1952	4000	3293
Dilithium5	EUF-CMA	5	2592	4864	4595
Falcon-512	EUF-CMA	1	897	1281	666
Falcon-1024	EUF-CMA	5	1793	2305	1280
SPHINCS+-SHA2-128f-simple	EUF-CMA	1	32	64	17088
SPHINCS+-SHA2-128s-simple	EUF-CMA	1	32	64	7856
SPHINCS+-SHA2-192f-simple	EUF-CMA	3	48	96	35664
SPHINCS+-SHAKE-128f-simple	EUF-CMA	1	32	64	17088

Table 2: Parameter set summary of Signature schemes.

embedded platforms. Findings indicate that Kyber outperforms Elliptic Curve Cryptography (ECC) in key exchange on all platforms, while SPHINCS+ is generally slower, especially in signing. It is suggested that this can be improved by integrating hardware support for hash function calculations. Sikeridis et al. [17] presents a detailed evaluation of the performance of NIST's post-quantum (PQ) signature algorithm candidates in the context of TLS 1.3 connection establishment under realistic network conditions. The results demonstrate that the adoption of at least two PQ signature algorithms would be viable with little additional overhead over current signature algorithms and that the PQ algorithms with the best performance for time-sensitive applications are Dilithium and Falcon. Furthermore, combining different PQ signature algorithms in a certificate chain can improve the overall handshake speed and throughput. They anticipates that the size of the signature and key will have the most significant impact on the handshake. Paquin et al. [15] introduced a framework for evaluating the performance of post-quantum cryptographic primitives in Transport Layer Security (TLS) under various network conditions, using the networking features of the Linux kernel. This study revealed that packet loss rates above 3-5% significantly affect post-quantum algorithms that fragment across many packets, and network latency tends to largely conceal the impact from algorithms with slower computations. Tasopoulos et al. [20] evaluated the performance of the post-quantum variant of TLS 1.3, including its execution time, memory, and bandwidth requirements, on resource-constrained IoT-supportive embedded devices, particularly the ARM Cortex-M4 platform. This represents the first comprehensive evaluation of PQ TLS 1.3 for all NIST PQC selections and upcoming candidates, specifically targeting resource-constrained embedded systems.

3 Simulator for Evaluating Post-Quantum Cryptography in TLS 1.3

This chapter details the simulator implementation used in assessing quantum-resistant cryptographic algorithms within the TLS 1.3 environment. Such simulators enable a controlled

setting for evaluating algorithm performance under varied network conditions. By replicating real-world scenarios, simulators illuminate the effects of cryptographic methods on key performance indicators like latency and throughput. These tools also assist in the scalability testing and final validation, thus minimizing risks during real-world deployments.

Quantum Safe OpenSSL The rise of quantum computing necessitates a reevaluation of existing cryptographic methods. In response, the Open Quantum Safe (OQS) project, an open-source initiative, was established. It focuses on developing and integrating quantum-resistant cryptographic algorithms into the prevalent network infrastructure. The OQS project has championed multiple quantum-resistant algorithms, notably through liboqs, a C library that provides quantum-safe encryption, key exchange, and digital signature functionalities. Alongside, OQS offers an OpenSSL fork, seamlessly introducing quantum-safe cryptography to prevailing software. Our simulator is built upon the OQS OpenSSL library. The OQS project provides liboqs, a C library offering quantum-safe public-key encryption, key exchange, and digital signature algorithms. Alongside liboqs, OQS also provides a fork of OpenSSL that integrates quantum-safe algorithms with OpenSSL's API. This enables existing software to easily use quantum-safe cryptography. We use the OQS OpenSSL library from the OQS project to implement a simulator of TLS1.3. Figure 1 is a block diagram.

NS-3 NS-3 is a discrete-event network simulator, widely used in both academia and industry for research and development. It's used to replicate real network environments in order to analyze protocols and applications, as well as network behaviors under various conditions. It allows for real-time simulations of network behavior under multiple scenarios, thereby enabling researchers to understand how these systems would function under a wide range of conditions. An integral feature of NS-3 lies in its modular design. This design approach enables users to selectively include or exclude different components, thereby facilitating the accurate reproduction of diverse network environments and conditions. NS-3 supports network protocols across various layers, providing detailed simulations of network layers, including those that feature the TCP/IP protocol stack. Moreover, it offers comprehensive support for network layers such as the physical layer, link layer, network layer, and transport layer.

3.1 simulator Implementation

We simulate client-server network communication using the NS3 simulator with Transport Layer Security (TLS) encryption implemented via the Oqs OpenSSL library. It is configured as shown in Figure 1. The 'SimpleTLSClient' represents a client that can set server details, initiate connections, send packets, and handle received data. Similarly, the 'SimpleTLSServer' acts as a server that can set port details, accept connections, and handle incoming data. The 'handshake' comprises functions necessary for establishing a TLS context and performing the SSL handshake, a procedure crucial for initiating a secure connection. It is written in C++ based on NS3 network simulator and OQS openssl.

3.1.1 SimpleTLSClient

The SimpleTLSClient class, designed within the NS3 simulation environment, integrates with the Transport Layer Security (TLS) using the OQS OpenSSL library. Upon instantiation, it initializes an empty socket and server port. The SetServerAddress function configures the server's IP and port. StartApplication initiates the client's socket connection to the server. Connect, upon receiving an InetSocketAddress, sets a data reception callback through HandleRead.

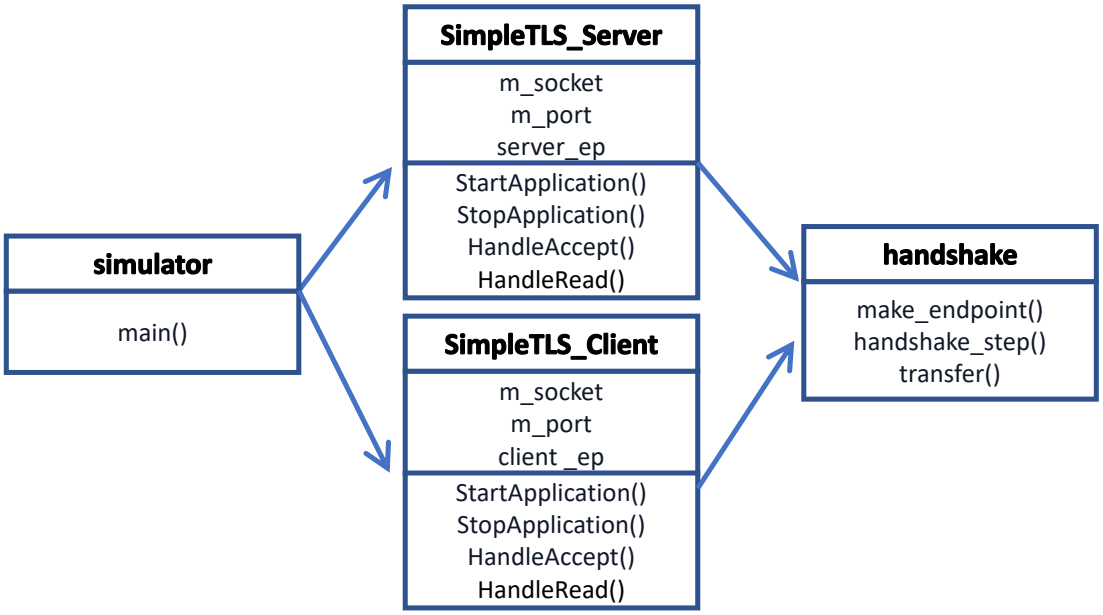

Figure 1: block diagram of the simulator.

KEM	skeygen/s	encap/s	decap/s	handshake/s
kyber512	108423	90331.6	129715.8	3572.76
kyber768	70927	62082.6	83326.9	3378.92
kyber1024	53706	45935.9	58828.8	3374.82
bikel1	4032	25222.5	1037.2	600.44
bikel3	1578	10848.5	297.9	208.99
hqc128	27970	14180.1	7911.1	2129.11
hqc192	12009	6112.9	3714.9	1373.07
hqc256	6580	3457.6	2058	942.06

Table 3: KEM OpenSSL (openssl speed) and TLS handshake performance (openssl s_time) signature algorithm selected as dilithium2.

Within SendPacket, a TLS connection is established using make_endpoint and handshake_step, followed by data reading and transmission to the server. HandleRead processes incoming data and, if the SSL handshake remains incomplete, continues the handshake and manages any pending data. StopApplication terminates the socket connection. This class embodies the process of securely establishing, sending, and receiving data between a client and server using TLS.

3.1.2 SimpleTLSServer

The ‘SimpleTLSServer‘ class, integrated with NS3 and TLS via OpenSSL, serves as the server component within a simulated client-server network model. Upon instantiation, an empty socket is initialized and a port is assigned using the ‘SetPort‘ method. The ‘StartApplication‘ method initializes the socket, binds it to the designated port, and prepares it to receive connections. Concurrently, it assigns the ‘HandleAccept‘ callback, which, upon a connection’s acceptance,

sign	sign/s	verify/s	handshake/s
dilithium2	14087.5	38544.0	3441.29
dilithium3	8704.3	22878.6	3056.64
dilithium5	7079.0	14414.5	2684.57
falcon512	4014.6	23451.8	3151.40
falcon1024	1968.0	12095.3	2580.10
sphincssha2128fsimple	134.7	1615.6	889.53
sphincssha2128ssimple	6.6	4079.3	1507.69
sphincsshake128fsimple	69.8	959.3	662.83
sphincssha2192fsimple	77.8	1125.5	674.56

Table 4: Signature OpenSSL (openssl speed) and TLS handshake performance (openssl s_time) KEM algorithm selected as kyber512.

initializes OpenSSL and sets up a secure TLS endpoint. Incoming data is processed through the 'HandleRead' method. If the SSL handshake is incomplete, it proceeds to the next handshake step, ensuring data integrity. The class concludes its operation with the 'StopApplication' method, which closes the listening socket. Collectively, 'SimpleTLSServer' encompasses the server's tasks: establishing secure connections, processing incoming data, and managing the SSL handshake.

3.1.3 handshake

The 'handshake' process facilitates the creation and management of TLS endpoints, representing the cryptographic parameters and state for a TLS connection. The module introduces several critical functions: - 'make_endpoint': Initializes a 'struct Endpoint' representing a TLS connection end. Depending on its role as a server or client, this function configures the context using OpenSSL functions, sets the minimum protocol version to TLS 1.3, and designates the key exchange algorithm. It further establishes SSL objects, memory BIOs (an OpenSSL I/O abstraction), and the corresponding state. - 'ssl_wants_rw': Inspects SSL error returns to determine if an operation should retry upon transport layer's readiness for read/write operations. - 'handshake_step': Executes a segment of the TLS handshake. If the handshake is complete, it confirms; otherwise, it calls 'SSL_do_handshake' to address any outstanding issues. This function's design supports incremental handshake steps, vital for simulations with non-blocking I/O contexts. 'transfer': Processes data from an endpoint's output BIO and dispatches it through the relevant socket, mimicking data transportation over the network. Together, these functions adeptly manage TLS connections, enabling the initiation and incremental handling of TLS handshakes—a pivotal aspect for realistic network simulations with asynchronous events and non-blocking I/O.

4 Evaluation

We carried out experiments on the algorithms chosen from the NIST PQC competition and the 4th round candidates. For key exchange algorithms, we selected Kyber, BIKE, and HQC. We excluded Classic McEliece due to its substantial key size. The signature algorithms chosen were Dilithium, Falcon, and SPHINCS+. We employed the library from OQS OpenSSL for our experiments. Our assumption was based on a standard web scenario, where only the server is

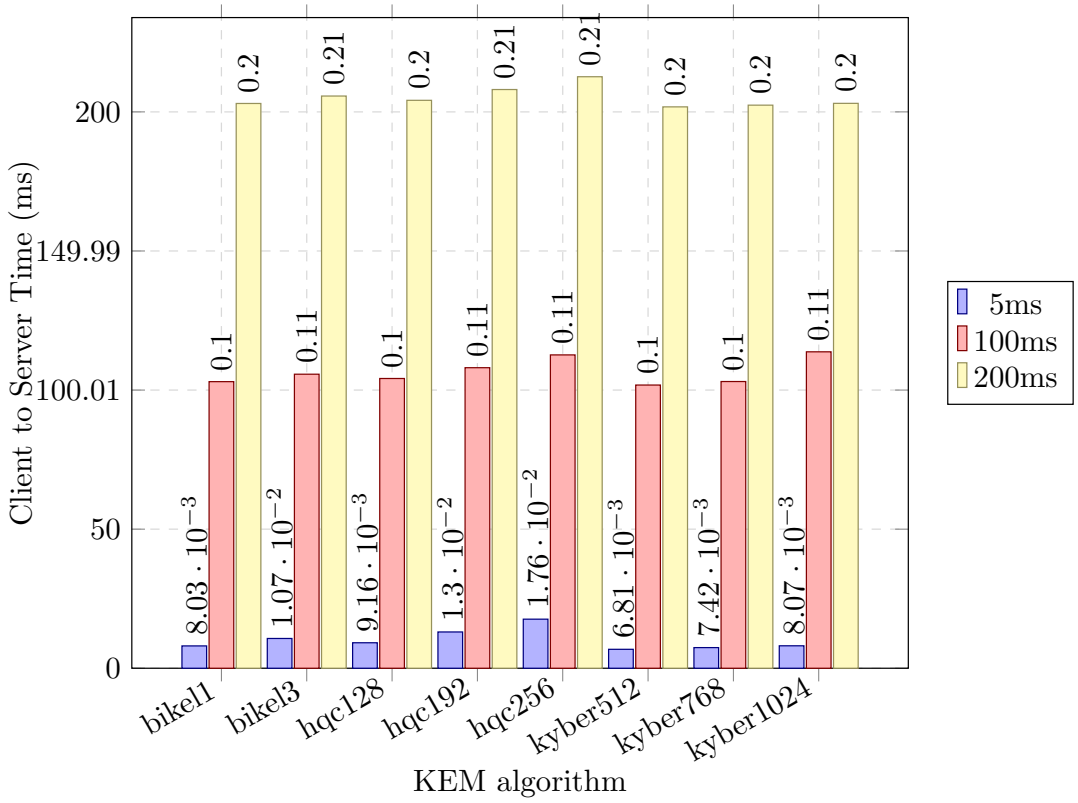

Figure 2: Results of time measurements from clienthello transmission to serverhello transmission for each key exchange algorithm a point-to-point network based on latency of 5 ms, 100 ms, and 200 ms.

authenticated using an X.509 certificate, with the TLS 1.3 handshake operating in a 1-round trip time (1-RTT) mode. A single local machine was used for experimentation, equipped with an Intel i5-8259U 2.30GHz processor, 16GB RAM, and Ubuntu 20.04 was the chosen environment. We carried out two types of network simulations: point-to-point and Wi-Fi. In both setups, a pair of nodes emulated a client-server relationship, with internet protocols and global routing set up for packet exchange.

4.1 Local testing using OpenSSL

The results of local tests utilizing OpenSSL are demonstrated in Tables 3 and 4. In the former, the Kyber algorithm (512, 768, 1024) outperforms others. However, its performance sees a gradual decline as the Kyber version increases. BIKE (l1, l3) and HQC (128, 192, 256) show relatively lower performance. Among the KEM algorithms, kyber512 exhibits the top handshake performance (3572.76 handshake/s), trailed by kyber768 (3378.92 handshake/s) and kyber1024 (3374.82 handshake/s). The hqc256 algorithm recorded the lowest handshake performance at 942.06 handshake/s. Table 4 reveals that dilithium2, 3, 5, and falcon512, 1024 possess high signature generation and verification capabilities. However, even among these, performance declines as the version scales up. SPHINCS algorithms consistently exhibit the lowest perfor-

mance. Here, Dilithium2 registers the highest performance (3441.29 handshake/s), followed by falcon512 (3151.40 handshake/s) and dilithium3 (3056.64 handshake/s). The algorithm with the lowest handshake performance in this set is sphincssha2128fsimple at 889.53 handshake/s. In the local tests using OpenSSL, the combination of kyber512 KEM and dilithium2 signature algorithms yielded the best handshake performance.

4.2 Point-to-Point Network Simulation

In the simulation, a pair of nodes are instantiated within a NodeContainer object named "nodes." Using PointToPointHelper, a connection is established with a defined data rate and delay. The network introduces a delay model with a DataRate object set to "5Mbps." The PcapFileWrapper assists in tracking and storing packet-level data. InternetStackHelper aids in installing internet protocols on the nodes, which are then assigned IPv4 addresses from the subnet 10.1.1.0/24. Ipv4GlobalRoutingHelper::PopulateRoutingTables() is used for global routing configuration. One node (nodes.Get(1)) runs the SimpleTLSServer application, while the counterpart (nodes.Get(0)) operates the SimpleTLSClient application. This arrangement facilitates secure client-server communication over a TLS connection. The server application listens on port 4433, with defined start and stop times for both applications. Each device captures packets to a PCAP file using the wifiPhy.EnablePcap() function. Tables 7, 8, and 9 depict the handshake measurements for each post-quantum cryptography (PQC) algorithm.

4.2.1 Difference between handshake steps

The time taken from sending the client hello to sending the server hello (client_to_server_times) remains roughly constant across different KEM and signature schemes at any given network delay. This suggests that the choice of signature scheme doesn't significantly impact this part of the communication, as there's no signature process involved in this phase. However, the time from sending the server hello to the finished message (server_to_finished_times) varies considerably across different signature and KEM schemes. This indicates that the choice of signature and KEM schemes significantly affects the server hello send client finish send process. Several algorithm combinations (dilithium2-hqc192, dilithium2-hqc256, dilithium3-hqc192, dilithium3-hqc256, dilithium5-bikel3, dilithium5-hqc128, dilithium5-hqc192, dilithium5-hqc256, falcon512-hqc256, falcon1024-hqc 256, SPHINCS+ variants) increasing network delay is more pronounced in the server_to_finished_times compared to client_to_server_times.

4.2.2 Signature schemes

Larger key sizes and ciphertext sizes generally correspond to longer transmission and processing times. For example, Dilithium5 and Falcon1024, which have larger key sizes, tend to have longer times than Dilithium2 and Falcon512. However, key size and ciphertext size are not the only factors affecting transmission and processing time. Other factors, such as the efficiency of the algorithm, also play a significant role. This is evident in the performance of SPHINCS variants, which have the longest times despite not having the largest key or ciphertext sizes. Falcon512 and Falcon1024 usually perform better than Dilithium in terms of time, with Falcon512 generally performing better than Falcon1024. This differs from previous local testing. This seems to be influenced by the public key size and signature size rather than the speed performance of the algorithm due to the limitation of MSS(maximum segment size). SPHINCS variants perform the worst, with the highest times across all metrics, especially noticeable with SPHINCS-SHA2-192f-simple.

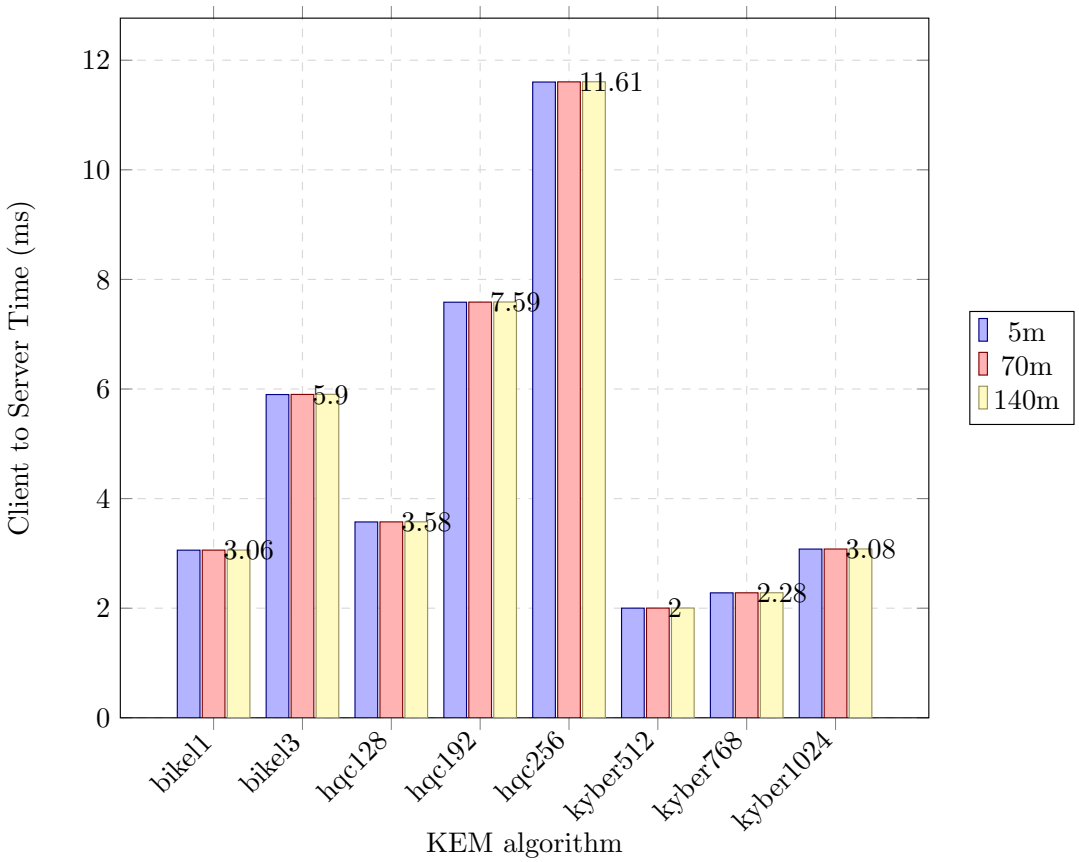

Figure 3: Results of time measurements from clienthello transmission to serverhello transmission for each key exchange algorithm in a point-to-point network based on distance of 5 m, 70 m, and 140 m.

4.2.3 Key encapsulation mechanisms

KYBER512 generally performs best, with the lowest times across almost all signature schemes and network delays. HQC192 and HQC256 tend to perform the worst, showing significantly increased times across most signature schemes. In the case of bike, it shows much better performance when compared to handshake performance in the local environment. In table 3, bike showed the worst performance, but it showed better performance than hqc128 considering the network environment. This seems to be the same effect of MSS in the signature.

4.2.4 Hybrid key exchange comparison

In the face of advancing quantum computing, hybrid key exchange mechanisms are essential. They merge traditional cryptographic methods with emerging post-quantum solutions, ensuring robust security. In this experiment, we tested combinations of traditional ECC (Elliptic Curve Cryptography) curves, including p256, p384, and p521, with post-quantum signature and KEM (Key Encapsulation Mechanism) algorithms.

The measurement results presented in Table 5 reveal difference among the signature algorithms.The p256_falcon512 algorithm consistently displayed the fastest server response time across all delay scenarios. In comparison, p256_dilithium2 and p384_dilithium3 showed moderate server response times, especially as the delay increased. SPHINCS variants, especially p256_sphincssha2128fsimple and p256_sphincsshake128fsimple, recorded the longest server response times across the board. This observation aligns with previous findings that SPHINCS algorithms generally have lower performance. Interestingly, algorithms combined with higher ECC curves, such as p521 (e.g., p521_dilithium5 and p521_falcon1024), showed relatively good performance, with p521_falcon1024 being notably efficient.

Turning to the KEM algorithms in Table 6, it is evident that the p256_kyber512 algorithm outperforms others, demonstrating the quickest server response time across all delay scenarios.This finding confirms the strength of KYBER in handshake performances. In contrast, the p256_bikel1 and p256_hqc128 algorithms showed slightly increased server response times, especially as the delay increased. Algorithms paired with the p384 curve, such as p384_bikel3 and p384_hqc192, displayed moderate performance. However, p384_kyber768 maintained a relatively consistent and efficient server response time. When paired with the p521 curve, the p521_hqc256 algorithm showcased the most significant delay, especially in scenarios with a 200ms latency. On the other hand, p521_kyber1024 remained consistent and efficient.

In conclusion, while hybrid key exchange mechanisms offer an added layer of security, the choice of post-quantum algorithm plays a pivotal role in determining performance. Algorithms like KYBER and Falcon consistently outperform others, such as SPHINCS and HQC, in hybrid settings. Thus, judicious selection of the combination of traditional and post-quantum algorithms is crucial to strike a balance between security and performance in real-world scenarios.

SIG	5ms delay		100ms delay		200ms delay	
	C→S (ms)	S→F (ms)	C→S (ms)	S→F (ms)	C→S (ms)	S→F (ms)
p256_falcon512	6.806	11.472	101.806	106.478	201.806	206.48
p256_dilithium2	6.806	20.162	101.806	305.163	201.806	605.077
p256_sphincssha2128fsimple	6.806	69.606	101.806	706.190	201.806	1212.728
p256_sphincssha2128ssimple	6.806	37.053	101.806	506.256	201.806	1005.234
p256_sphincsshake128fsimple	6.806	69.605	101.806	706.189	201.806	1212.728
p384_dilithium3	6.806	24.475	101.806	309.478	201.806	609.389
p384_sphincssha2192fsimple	6.806	135.35	101.806	905.678	201.806	1468.283
p521_dilithium5	6.806	30.355	101.806	315.357	201.806	615.443
p521_falcon1024	6.806	17.227	101.806	302.221	201.806	602.323

Table 5: Results of handshake time measurements for each SIG algorithm in a point-to-point network based on latency of 5 ms, 100 ms, and 200 ms.

4.3 Wi-Fi network Simulation

In the simulator, two nodes are created within a `NodeContainer` object named nodes. Following this, a Wi-Fi network is configured for the two nodes with `WifiMacHelper` and `YansWifiPhyHelper` objects, which handle the MAC and PHY layers of the Wi-Fi devices respectively. These nodes operate in ad-hoc mode (`ns3::AdhocWifiMac`) forming a peer-to-peer network without a central access point. The Wi-Fi channel employs a constant speed propagation delay model and a Friis propagation loss model, with the `WifiHelper` object assisting in installing the Wi-Fi devices on the nodes using the 802.11b standard. `GridPositionAllocator` is used to assign fixed positions to the nodes on a grid, maintaining a spacing in the x-direction. Stationary conditions throughout the simulation are ensured

KEM	5ms delay		100ms delay		200ms delay	
	C→S (ms)	S→F (ms)	C→S (ms)	S→F (ms)	C→S (ms)	S→F (ms)
p256_bikel1	8.268	21.226	103.268	306.226	203.268	606.312
p256_hqc128	9.574	26.397	104.574	311.397	204.574	611.483
p256_kyber512	7.083	19.937	102.083	304.937	202.083	604.851
p384_bikel3	11.132	24.09	106.132	309.09	206.132	609.004
p384_hqc192	13.608	34.411	108.608	503.612	208.608	811.088
p384_kyber768	7.083	19.937	102.748	305.501	202.748	605.415
p521_hqc256	20.217	45.063	305.217	513.33	605.131	1013.329
p521_kyber1024	7.083	19.937	103.42	306.327	203.42	606.413

Table 6: Results of handshake time measurements for each KEM algorithm in a point-to-point network based on latency of 5 ms, 100 ms, and 200 ms.

by the `ConstantPositionMobilityModel`. To handle internet protocols, the stack is installed on the nodes using the `InternetStackHelper`, with each node being assigned an IPv4 address from the subnet 10.1.1.0/24 via the `Ipv4AddressHelper`. Global routing is configured with `PopulateRoutingTables()`, enabling each node to route to any other node in the network. The second node (`nodes.Get(1)`) is installed with a `SimpleTLSServer` application, and the first node (`nodes.Get(0)`) with a `SimpleTLSClient`, facilitating a secure client-server interaction over a TLS connection. The server listens on port 4433, with specified start and stop times for both applications. Finally, each device on the nodes is set to capture packets to a PCAP file using the `wifiPhy.EnablePcap()` function. Tables 10, 11, 12 are handshake measurement results of each pqc algorithm. We measured at short distance of 5m, medium distance of 70m, and maximum distance of 140m. Within this range, the delay was not significant, so there was no difference even at the maximum distance. Overall, the results were similar to the measurements in Point to Point.

5 Conclusion

This study evaluated the influence of quantum-resistant cryptography on TLS 1.3 across diverse network conditions. Emphasizing latency and bandwidth constraints, simulations were executed in point-to-point and Wi-Fi configurations, elucidating TLS 1.3's behavior in varied setups. Our research addressed extant gaps in the understanding of post-quantum cryptography in TLS. The controlled simulation environment was pivotal in affording insights otherwise obfuscated in real-world networks, thus providing a robust performance appraisal of TLS 1.3 with quantum-resistant encryption. The assessment incorporated multiple post-quantum cryptographic algorithms from the NIST PQC competition, including the 4th round candidates. Key exchange algorithms—Kyber, BIKE, and HQC—and signature algorithms—Dilithium, Falcon, and SPHINCS+—were critically examined. Simulations underscored the efficiency disparities of these algorithms across network environments. Our findings accentuated that signature and key encapsulation choices markedly influence TLS handshake phases. Parameters like key and ciphertext sizes, coupled with inherent algorithmic efficiency, dictate transmission and processing durations. Notably, KYBER512 exhibited optimal performance, HQC variants reflected elevated durations, Falcon outperformed Dilithium, and SPHINCS variants were most protracted. In Wi-Fi scenarios, observed trends largely mirrored point-to-point settings with network delay impacts being inconsequential, attributable to the confined measurement spectrum. Our scope was restricted to NIST PQC competition algorithms and 4th round candidates. Given

the nascent stage of PQC algorithms and extant regulatory mandates such as FIPS, a hybrid approach to TLS is prevalent. Subsequent endeavors will target performance assessments of these hybrid solutions, broadening the scope to diverse NS3-supported network settings and energy consumption facets. The objective remains to holistically comprehend the practical ramifications of migrating to post-quantum cryptography.

References

[1] Gorjan Alagic, Daniel Apon, David Cooper, Quynh Dang, Thinh Dang, John Kelsey, Jacob Lichtinger, Carl Miller, Dustin Moody, Rene Peralta, et al. Status report on the third round of the nist post-quantum cryptography standardization process. *US Department of Commerce, NIST*, 2022.

[2] Nicolas Aragon, Paulo Barreto, Slim Bettaieb, Loic Bidoux, Olivier Blazy, Jean-Christophe Deneuville, Phillipe Gaborit, Shay Gueron, Tim Guneysu, Carlos Aguilar Melchor, et al. Bike. *Round 2 submission to the NIST PQC Project*, 2019.

[3] William Barker, William Polk, and Murugiah Souppaya. Getting ready for post-quantum cryptography: explore challenges associated with adoption and use of post-quantum cryptographic algorithms. *The Publications of NIST Cyber Security White Paper (DRAFT), CSRC, NIST, GOV*, 26, 2020.

[4] Daniel J Bernstein, Tung Chou, Tanja Lange, Ingo von Maurich, Rafael Misoczki, Ruben Niederhagen, Edoardo Persichetti, Christiane Peters, Peter Schwabe, Nicolas Sendrier, et al. Classic mceliece: conservative code-based cryptography. *NIST submissions*, 2017.

[5] Daniel J. Bernstein, Christoph Dobraunig, Maria Eichlseder, Scott R. Fluhrer, Stefan-Lukas Gazdag, Andreas Hülsing, Panos Kampanakis, Stefan Kölbl, Tanja Lange, Martin M. Lauridsen, Florian Mendel, Ruben Niederhagen, Christian Rechberger, Joost Rijneveld, and Peter Schwabe. Sphincs + submission to the nist post-quantum project. 2017.

[6] Joppe Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M Schanck, Peter Schwabe, Gregor Seiler, and Damien Stehlé. Crystals-kyber: a cca-secure module-lattice-based kem. In *2018 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 353–367. IEEE, 2018.

[7] Kevin Bürstinghaus-Steinbach, Christoph Krauß, Ruben Niederhagen, and Michael Schneider. Post-quantum tls on embedded systems: Integrating and evaluating kyber and sphincs+ with mbed tls. In *Proceedings of the 15th ACM Asia Conference on Computer and Communications Security*, pages 841–852, 2020.

[8] Wouter Castryck and Thomas Decru. An efficient key recovery attack on sidh. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 423–447. Springer, 2023.

[9] Eric Crockett, Christian Paquin, and Douglas Stebila. Prototyping post-quantum and hybrid key exchange and authentication in tls and ssh. *Cryptology ePrint Archive*, 2019.

[10] Léo Ducas, Eike Kiltz, Tancrede Lepoint, Vadim Lyubashevsky, Peter Schwabe, Gregor Seiler, and Damien Stehlé. Crystals-dilithium: A lattice-based digital signature scheme. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 238–268, 2018.

[11] Pierre-Alain Fouque, Jeffrey Hoffstein, Paul Kirchner, Vadim Lyubashevsky, Thomas Pornin, Thomas Prest, Thomas Ricosset, Gregor Seiler, William Whyte, Zhenfei Zhang, et al. Falcon: Fast-fourier lattice-based compact signatures over ntru. *Submission to the NIST's post-quantum cryptography standardization process*, 36(5), 2018.

[12] David Jao, Reza Azarderakhsh, Matthew Campagna, Craig Costello, Luca De Feo, Basil Hess, Amir Jalali, Brian Koziel, Brian LaMacchia, Patrick Longa, et al. Supersingular isogeny key encapsulation. *Submission to the NIST Post-Quantum Standardization Project*, 2017.

[13] David Joseph, Rafael Misoczki, Marc Manzano, Joe Tricot, Fernando Dominguez Pinuaga, Olivier Lacombe, Stefan Leichenauer, Jack Hidary, Phil Venables, and Royal Hansen. Transitioning organizations to post-quantum cryptography. *Nature*, 605(7909):237–243, 2022.

[14] Carlos Aguilar Melchor, Nicolas Aragon, Slim Bettaieb, Loıc Bidoux, Olivier Blazy, Jean-Christophe Deneuville, Philippe Gaborit, Edoardo Persichetti, Gilles Zémor, and IC Bourges. Hamming quasi-cyclic (hqc). *NIST PQC Round*, 2(4):13, 2018.

[15] Christian Paquin, Douglas Stebila, and Goutam Tamvada. Benchmarking post-quantum cryptography in tls. In *Post-Quantum Cryptography: 11th International Conference, PQCrypto 2020, Paris, France, April 15–17, 2020, Proceedings 11*, pages 72–91. Springer, 2020.

[16] Peter W Shor. Algorithms for quantum computation: discrete logarithms and factoring. In *Proceedings 35th annual symposium on foundations of computer science*, pages 124–134. Ieee, 1994.

[17] Dimitrios Sikeridis, Panos Kampanakis, and Michael Devetsikiotis. Post-quantum authentication in tls 1.3: a performance study. *Cryptology ePrint Archive*, 2020.

[18] Douglas Stebila, Scott Fluhrer, and Shay Gueron. Hybrid key exchange in tls 1.3. *IETF draft*, 2020.

[19] Douglas Stebila and Michele Mosca. Post-quantum key exchange for the internet and the open quantum safe project. In *International Conference on Selected Areas in Cryptography*, pages 14–37. Springer, 2016.

[20] George Tasopoulos, Jinhui Li, Apostolos P Fournaris, Raymond K Zhao, Amin Sakzad, and Ron Steinfeld. Performance evaluation of post-quantum tls 1.3 on resource-constrained embedded systems. In *Information Security Practice and Experience: 17th International Conference, ISPEC 2022, Taipei, Taiwan, November 23–25, 2022, Proceedings*, pages 432–451. Springer, 2022.

Appendices

SIG	KEM	5ms delay		100ms delay		200ms delay	
		C→S(ms)	S→F(ms)	C→S(ms)	S→F(ms)	C→S(ms)	S→F(ms)
dilithium2	bikel1	8.027	18.757	103.027	113.757	203.027	213.757
dilithium2	bikel3	10.702	21.432	105.702	116.432	205.702	216.432
dilithium2	hqc128	9.16	23.652	104.16	118.652	204.16	218.652
dilithium2	hqc192	13.04	31.376	108.04	304.344	208.04	604.257
dilithium2	hqc256	17.64	40.571	112.64	313.539	212.64	613.452
dilithium2	kyber512	6.806	17.469	101.806	112.469	201.806	212.469
dilithium2	kyber768	7.42	17.981	102.42	112.981	202.42	212.981
dilithium2	kyber1024	8.07	18.749	103.07	113.749	203.07	213.749
dilithium3	bikel1	8.027	22.817	103.027	117.817	203.027	217.817
dilithium3	bikel3	10.702	25.493	105.702	120.493	205.702	220.493
dilithium3	hqc128	9.16	27.713	104.16	122.713	204.16	222.713
dilithium3	hqc192	13.04	35.436	108.04	308.404	208.04	608.318
dilithium3	hqc256	17.64	44.632	112.64	317.6	212.64	617.513
dilithium3	kyber512	6.806	21.494	101.806	116.494	201.806	216.494
dilithium3	kyber768	7.42	22.007	102.42	117.007	202.42	217.007
dilithium3	kyber1024	8.07	22.81	103.07	117.81	203.07	217.81
dilithium5	bikel1	8.027	28.251	103.027	123.251	203.027	223.251
dilithium5	bikel3	10.702	30.926	105.702	303.894	205.702	603.808
dilithium5	hqc128	9.16	33.147	104.16	306.115	204.16	606.201
dilithium5	hqc192	13.04	40.87	108.04	313.838	208.04	613.752
dilithium5	hqc256	17.64	50.065	112.64	323.033	212.64	622.947
dilithium5	kyber512	6.806	26.928	101.806	121.928	201.806	221.928
dilithium5	kyber768	7.42	27.476	102.42	122.476	202.42	222.476
dilithium5	kyber1024	8.07	28.243	103.07	123.243	203.07	223.243

Table 7: Results of handshake time measurements for each Dilithium algorithm in a point-to-point network based on latency of 5 ms, 100 ms, and 200 ms.

SIG	KEM	5ms delay		100ms delay		200ms delay	
		C→S(ms)	S→F(ms)	C→S(ms)	S→F(ms)	C→S(ms)	S→F(ms)
falcon512	bikel1	8.027	12.134	103.027	107.134	203.027	207.136
falcon512	bikel3	10.702	14.642	105.702	109.634	205.702	209.642
falcon512	hqc128	9.16	17.035	104.16	112.038	204.16	212.035
falcon512	hqc192	13.04	24.588	108.04	119.58	208.04	219.579
falcon512	hqc256	17.64	33.774	112.64	306.744	212.64	606.832
falcon512	kyber512	6.806	10.84	101.806	105.845	201.806	205.846
falcon512	kyber768	7.42	11.365	102.42	106.36	202.42	206.354
falcon512	kyber1024	8.07	12.125	103.07	107.126	203.07	207.13
falcon1024	bikel1	8.027	15.779	103.027	110.776	203.027	210.782
falcon1024	bikel3	10.702	18.28	105.702	113.277	205.702	213.29
falcon1024	hqc128	9.16	20.675	104.16	115.673	204.16	215.672
falcon1024	hqc192	13.04	28.219	108.04	123.23	208.04	223.228
falcon1024	hqc256	17.64	37.425	112.64	310.401	212.64	610.478
falcon1024	kyber512	6.806	14.283	101.806	109.285	201.806	209.285
falcon1024	kyber768	7.42	14.824	102.42	110.005	202.42	210.005
falcon1024	kyber1024	8.07	15.77	103.07	110.771	203.07	210.765

Table 8: Results of handshake time measurements for each Falcon algorithm in a point-to-point network based on latency of 5 ms, 100 ms, and 200 ms.

SIG	KEM	5ms delay		100ms delay		200ms delay	
		C→S(ms)	S→F(ms)	C→S(ms)	S→F(ms)	C→S(ms)	S→F(ms)
sphincssha2128fsimple	bikel1	8.027	66.081	103.027	339.049	203.027	638.963
sphincssha2128fsimple	bikel3	10.702	68.584	105.702	341.552	205.702	641.638
sphincssha2128fsimple	hqc128	9.16	70.977	104.16	343.945	204.16	643.859
sphincssha2128fsimple	hqc192	13.04	78.528	108.04	509.803	208.04	829.464
sphincssha2128fsimple	hqc256	17.64	87.723	112.64	518.998	212.64	1016.627
sphincssha2128fsimple	kyber512	6.806	64.794	101.806	337.762	201.806	637.675
sphincssha2128fsimple	kyber768	7.42	65.306	102.42	338.274	202.42	638.188
sphincssha2128fsimple	kyber1024	8.07	66.074	103.07	339.042	103.07	339.042
sphincssha2128ssimple	bikel1	8.027	35.009	103.027	307.977	203.027	607.891
sphincssha2128ssimple	bikel3	10.702	37.512	105.702	310.48	205.702	610.566
sphincssha2128ssimple	hqc128	9.16	39.905	104.16	312.873	204.16	612.787
sphincssha2128ssimple	hqc192	13.04	47.456	108.04	320.424	208.04	620.51
sphincssha2128ssimple	hqc256	17.64	56.616	112.64	329.584	212.64	629.497
sphincssha2128ssimple	kyber512	6.806	33.514	101.806	306.482	201.806	606.568
sphincssha2128ssimple	kyber768	7.42	34.026	102.42	306.994	202.42	607.08
sphincssha2128ssimple	kyber1024	8.07	35.002	103.07	307.97	203.07	607.883
sphincsshake128fsimple	bikel1	8.027	66.081	103.027	339.049	203.027	638.963
sphincsshake128fsimple	bikel3	10.702	68.584	105.702	341.552	205.702	641.638
sphincsshake128fsimple	hqc128	9.16	70.977	104.16	343.945	204.16	643.859
sphincsshake128fsimple	hqc192	13.04	78.528	108.04	509.803	208.04	829.464
sphincsshake128fsimple	hqc256	17.64	87.723	112.64	518.998	212.64	1016.627
sphincsshake128fsimple	kyber512	6.806	64.794	101.806	337.762	201.806	637.675
sphincsshake128fsimple	kyber768	7.42	65.306	102.42	338.274	202.42	638.188
sphincsshake128fsimple	kyber1024	8.07	66.074	103.07	339.042	203.07	638.955
sphincssha2192fsimple	bikel1	8.027	128.782	103.027	560.057	203.027	1057.513
sphincssha2192fsimple	bikel3	10.702	131.285	105.702	562.56	205.702	1060.189
sphincssha2192fsimple	hqc128	9.16	133.678	104.16	564.953	204.16	1062.409
sphincssha2192fsimple	hqc192	13.04	141.228	108.04	572.504	208.04	1070.132
sphincssha2192fsimple	hqc256	17.64	150.424	112.64	581.699	212.64	1079.328
sphincssha2192fsimple	kyber512	6.806	127.286	101.806	558.562	201.806	1056.19
sphincssha2192fsimple	kyber768	7.42	127.799	102.42	559.074	202.42	1056.703
sphincssha2192fsimple	kyber1024	8.07	128.774	103.07	560.05	203.07	1057.506

Table 9: Results of handshake time measurements for each Sphincs+ algorithm in a point-to-point network based on latency of 5 ms, 100 ms, and 200 ms.

SIG	KEM	5m distance		70ms distance		140ms distance	
		C→S(ms)	S→F(ms)	C→S(ms)	S→F(ms)	C→S(ms)	S→F(ms)
dilithium2	bikel1	3.059	12.71	3.06	12.713	3.061	12.715
dilithium2	bikel3	5.899	15.429	5.901	15.432	5.903	15.436
dilithium2	hqc128	3.574	17.322	3.575	17.326	3.576	17.329
dilithium2	hqc192	7.584	23.943	7.586	23.948	7.588	23.953
dilithium2	hqc256	11.602	32.076	11.605	32.083	11.607	32.091
dilithium2	kyber512	2.001	12.124	2.002	12.127	2.003	12.129
dilithium2	kyber768	2.28	12.357	2.281	12.36	2.282	12.362
dilithium2	kyber1024	3.079	12.706	3.08	12.709	3.081	12.711
dilithium3	bikel1	3.059	16.942	3.06	16.946	3.061	16.949
dilithium3	bikel3	5.899	19.042	5.901	19.046	5.903	19.05
dilithium3	hqc128	3.574	22.093	3.575	22.098	3.576	22.102
dilithium3	hqc192	7.584	27.616	7.586	27.622	7.588	27.628
dilithium3	hqc256	11.602	36.169	11.605	36.177	11.607	36.186
dilithium3	kyber512	2.001	15.798	2.002	15.802	2.003	15.805
dilithium3	kyber768	2.28	16.031	2.281	16.035	2.282	16.038
dilithium3	kyber1024	3.079	16.939	3.08	16.943	3.081	16.946
dilithium5	bikel1	3.059	22.581	3.06	22.586	3.061	22.59
dilithium5	bikel3	5.899	23.959	5.901	23.964	5.903	23.969
dilithium5	hqc128	3.574	27.699	3.575	27.705	3.576	27.71
dilithium5	hqc192	7.584	32.472	7.586	32.479	7.588	32.486
dilithium5	hqc256	11.602	40.506	11.605	40.515	11.607	40.525
dilithium5	kyber512	2.001	20.054	2.002	20.058	2.003	20.063
dilithium5	kyber768	2.28	20.806	2.281	20.81	2.282	20.815
dilithium5	kyber1024	3.079	22.573	3.08	22.578	3.081	22.582

Table 10: Results of handshake time measurements for each Dilithium algorithm in a point-to-point network based on distance of 5 m, 70 m, and 140 m.

SIG	KEM	5ms distance		100ms distance		200ms distance	
		C→S(ms)	S→F(ms)	C→S(ms)	S→F(ms)	C→S(ms)	S→F(ms)
falcon512	bikel1	3.059	6.567	3.06	6.572	3.061	6.574
falcon512	bikel3	5.899	8.49	5.901	8.492	5.903	8.495
falcon512	hqc128	3.574	11.103	3.575	11.107	3.576	11.108
falcon512	hqc192	7.584	17.405	7.586	17.409	7.588	17.412
falcon512	hqc256	11.602	25.756	11.605	25.765	11.607	25.769
falcon512	kyber512	2.001	6.106	2.002	6.108	2.003	6.105
falcon512	kyber768	2.28	6.34	2.281	6.342	2.282	6.338
falcon512	kyber1024	3.079	6.567	3.08	6.57	3.081	6.57
falcon1024	bikel1	3.059	10.53	3.06	10.535	3.061	10.54
falcon1024	bikel3	5.899	12.816	5.901	12.82	5.903	12.821
falcon1024	hqc128	7.584	21.15	3.575	15.47	3.576	15.475
falcon1024	hqc192	7.588	21.155	7.586	21.151	7.588	21.155
falcon1024	hqc256	11.602	29.479	11.605	29.487	11.607	29.493
falcon1024	kyber512	2.001	8.228	2.002	8.232	2.003	8.235
falcon1024	kyber768	2.28	10.001	2.281	10.006	2.282	10.008
falcon1024	kyber1024	3.079	10.53	3.08	10.533	3.081	10.538

Table 11: Results of handshake time measurements for each Falcon algorithm in a point-to-point network based on distance of 5 m, 70 m, and 140 m.

SIG	KEM	5ms distance		100ms distance		200ms distance	
		C→S(ms)	S→F(ms)	C→S(ms)	S→F(ms)	C→S(ms)	S→F(ms)
sphincssha2128fsimple	bikel1	3.059	56.235	3.06	56.246	3.061	56.258
sphincssha2128fsimple	bikel3	5.899	56.148	5.901	56.159	5.903	56.17
sphincssha2128fsimple	hqc128	3.574	60.827	3.575	60.839	3.576	60.852
sphincssha2128fsimple	hqc192	7.584	64.861	7.586	64.874	7.588	64.887
sphincssha2128fsimple	hqc256	11.602	73.015	11.605	73.03	11.607	73.046
sphincssha2128fsimple	kyber512	2.001	55.73	2.002	55.741	2.003	55.753
sphincssha2128fsimple	kyber768	2.28	55.962	2.281	55.973	2.282	55.985
sphincssha2128fsimple	kyber1024	3.079	56.231	3.08	56.242	3.081	56.254
sphincssha2128ssimple	bikel1	3.059	29.85	3.06	29.857	3.061	29.863
sphincssha2128ssimple	bikel3	5.899	29.262	5.901	29.268	5.903	29.274
sphincssha2128ssimple	hqc128	3.574	34.042	3.575	34.05	3.576	34.057
sphincssha2128ssimple	hqc192	7.588	38.352	7.586	38.344	7.588	38.352
sphincssha2128ssimple	hqc256	11.602	45.851	11.605	45.86	11.607	45.87
sphincssha2128ssimple	kyber512	2.001	25.338	2.002	25.343	2.003	25.348
sphincssha2128ssimple	kyber768	2.28	25.571	2.281	25.576	3.081	29.859
sphincssha2128ssimple	kyber1024	3.079	29.846	3.08	29.853	3.08	29.853
sphincsshake128fsimple	bikel1	3.059	56.235	3.06	56.246	3.061	56.258
sphincsshake128fsimple	bikel3	5.899	56.148	5.901	56.159	5.903	56.17
sphincsshake128fsimple	hqc128	3.574	60.827	3.575	60.839	3.576	60.852
sphincsshake128fsimple	hqc192	7.584	64.861	7.586	64.874	7.588	64.887
sphincsshake128fsimple	hqc256	11.602	73.015	11.605	73.03	11.607	73.046
sphincsshake128fsimple	kyber512	2.001	55.73	2.002	55.741	2.003	55.753
sphincsshake128fsimple	kyber768	2.28	55.962	2.281	55.973	2.282	55.985
sphincsshake128fsimple	kyber1024	3.079	56.231	3.08	56.242	3.081	56.254
sphincssha2192fsimple	bikel1	3.059	114.273	3.06	114.296	3.061	114.321
sphincssha2192fsimple	bikel3	5.899	113.686	5.901	113.708	5.903	113.733
sphincssha2192fsimple	hqc128	3.574	118.065	3.575	118.089	3.576	118.114
sphincssha2192fsimple	hqc192	7.584	124.21	7.586	124.235	7.588	124.261
sphincssha2192fsimple	hqc256	11.602	134.929	11.605	134.956	11.607	134.986
sphincssha2192fsimple	kyber512	2.001	111.948	2.002	111.971	2.003	111.995
sphincssha2192fsimple	kyber768	2.28	112.181	2.281	112.204	2.282	112.228
sphincssha2192fsimple	kyber1024	3.079	114.269	3.08	114.292	3.081	114.317

Table 12: Results of handshake time measurements for each Sphincs+ algorithm in a point-to-point network based on distance of 5 m, 70 m, and 140 m.