Performance Evaluation of Quantum-Resistant TLS for Consumer IoT Devices

Jessica Bozhko*,Yacoub Hanna†, Ricardo Harrilal-Parchment†, Samet Tonyali‡, Kemal Akkaya†
*Dept. of Computer Science, Auburn University, Auburn, USA
Email: jab0245@auburn.edu
†Dept. of Electrical and Computer Engineering, Florida International University, Miami, USA
Email: {yhann002, rharr119, kakkaya}@fiu.edu
‡Dept. of Software Engineering, Gumushane University, Gumushane, Turkey
Email: samet.tonyali@gumushane.edu.tr

***Abstract*—Post-quantum (PQ) cryptographic algorithms are currently being developed to be able to resist attacks by quantum computers. The practical use of these algorithms for securing networks will depend on their computational and communication efficiency. In particular, this is critical for the security of wireless communications within the context of consumer IoT devices that may have limited computational power and depend on a constrained wireless bandwidth. To this end, there is a need to evaluate the performance of widely used application layer security standards such as transport layer security (TLS) to understand the use of the existing PQ algorithms that are being evaluated by NIST as a replacement to the current cryptographic algorithms. This paper focuses on two widely used IoT standards Bluetooth Low Energy (BLE) and WiFi to find out the optimal performing PQ algorithm for their security when used in end-to-end connections over the Internet. By implementing the capability for IP over BLE and all options of TLS connection establishment, we developed a client-server IoT testbed to measure the efficiency of PQ key encapsulation mechanisms (KEMs) and PQ digital signature algorithms. The test results showed that Kyber512 is the ideal KEM while Falcon-512 and Dilithium2 are the best signatures for BLE and WiFi devices. Based on this outcome, we developed a mechanism for IoT devices with multiple communication interfaces, that dynamically chooses a PQ KEM algorithm based on the MAC layer protocol being used at the time.**

***Index Terms*—Post-Quantum Cryptography; Key Encapsulation Mechanism; TLS; IoT; IP Over Bluetooth**

I. Introduction

Quantum computers are predicted to be exponentially more powerful than the computers in use today [1], which poses many threats to existing communications security standards [2]–[4]. For example, a quantum computer could be used to solve the elliptic curve discrete logarithm problem [5] and the integer factorization problem [6] on which security of elliptic curve cryptography (ECC) and RSA cryptosystem rely, respectively. These cryptographic schemes secure almost all electronic communications today, and any security breach in these schemes will result in the exposure of private and personal identifying information such as usernames, passwords, social security numbers, and credit card numbers. Fortunately, post-quantum cryptography (PQC) algorithms designed to resist attacks by quantum computers are already in the development and testing stages. For instance, The Open Quantum Safe (OQS) project [7] has an open source library of algorithms that claim to be able to provide levels of security specified by the National Institute of Standards and Technology (NIST) [8] that will be needed in a post-quantum world.

However, the application of these PQC algorithms still need further evaluation to understand their feasibility, performance, and overhead. One such area is the Internet of Things (IoT) where devices have restrictions in terms of computation and communication. As IoT is becoming a larger part of our daily lives, people are starting to own more devices whose communication relies on short-distance consumer wireless technologies such as Bluetooth (BLE) or WiFi.

As IoT devices typically supply data to remote servers, an end-to-end connection is becoming a de-facto use case that relies on the availability of IP addresses for the IoT devices to be reachable all over the world. Even in the case of BLE which is mostly known for device-to-device communication through pairing, (i.e., communication happens at the MAC layer), there are increasing number of consumer cases where BLE over IP will be needed (i.e., BLE device will use IP to connect to any remote server). For instance, BLE is used in a wide range of consumer applications in health, smart city, and smart home applications on devices such as heart rate sensors, security tags, and Fitbit. In each of these settings, the ability to be able to send data directly to remote servers is an important convenience for consumers. As an example, a Fitbit BLE device may use a smartphone as a gateway to send its measurements directly to a remote server through BLE over IP. As such, IETF has already came up with a new standard that will allow IPv6 over BLE [9]. This means in the future we may see WiFi routers supporting BLE routing capability to enable any BLE device to connect Internet through them without a need for a separate gateway.

However, since these IoT devices must securely communicate with each other and other remote servers, they use the same cryptographic standards as regular computers. When establishing end-to-end connections through IP, standards such as transport layer security protocol (TLS) or its UDP version DTLS utilize certain cryptographic algorithms such as RSA or ECC. This means that they will also be vulnerable in a post-quantum world that utilizes TLS for IoT-based wireless communications through BLE or WiFi. Therefore, it is impor-

tant to establish quantum-resistant cryptographic standards for securing all forms of communications for IoT devices.

To this end, in this paper, we evaluate the efficiency of PQC algorithms for consumer IoT devices when they establish a TLS connection which is the current standard on the Internet for any secure communication (i.e., https). In this setup, cryptography is not only used for signatures on the certificates but also for agreeing on a symmetric key. Therefore, a thorough evaluation is needed to measure the overhead when IP over BLE is started to be deployed, determine the best performing PQC algorithms and dynamically choose the best option based on the underlying wireless channel context.

For our evaluations, we chose to gather data on PQC algorithms currently considered by NIST [8] over a BLE or WiFi-based TLS connection. This end-to-end connection is created by a Raspberry Pi as a client and a personal laptop computer as a server. We measured the performance of both the key encapsulation mechanisms (KEMs) and the digital signature algorithms in TLS handshake and compared the performance of BLE and WiFi.

Experiment results showed that the underlying wireless connection used affects the performance significantly for both KEM and digital signatures. For instance, Kyber512 is an ideal KEM for BLE but for WiFi Dilithium2 can also be a viable option. Based on this result, we propose a dynamic KEM-switching mechanism within TLS in which an IoT client device determines what KEM algorithm it should choose that would be the most suitable for its underlying MAC layer.

This paper is organized as follows: In the next section, we summarize the related work. Section III provides some background on the used concepts while Section IV presents the IP over BLE model along with two TLS approaches. In Section V, we assess the performance, present a KEM-switching mechanism and evaluates its effects on the TLS handshake protocol. Lastly, Section VI concludes the paper.

II. Related Work

Some recent works also focus on the implementation of post-quantum algorithms over an IoT environment. The main work presented in this context [10] focuses on the performances of select PQ algorithms over a long-range connection and implements a similar approach in using two Raspberry Pi's to simulate IoT devices and to test the connection efficiency of PQ algorithms over long-range communications. Our goal in this work differs in that it deals with testing the efficiency of PQ algorithms for an end-to-end TLS connection that utilizes short-range wireless connections.

An empirical study [11] evaluates the efficiency of several PQ KEM and digital signature algorithms of the NIST's standardization process. The paper considers an IoT environment composed of a Raspberry Pi and an AWS server. In the experiments, the Raspberry Pi and the AWS server exchange messages to establish a TLS connection using Ethernet. Latency and message overhead of TLS handshakes are used as performance metrics. Our work differs from this study in the technology stacks (i.e., BLE) that we use.

A comprehensive study [12] integrates and evaluates Kyber and SPHINCS+ on embedded systems using mbed TLS. First, runtime of key generation, encryption, decryption, signing, and verification operations for the algorithms is measured on the platforms. Then, the algorithms are integrated into mbed TLS library, and TLS handshake performance of the platforms is measured in terms of runtime and message overhead. However, the platforms operate on either Ethernet, WiFi, or both, but none of them supports Bluetooth which is a very common IoT network connectivity technology. Also, the study considers only Kyber and SPHINCS+ whereas our study focuses on multiple PQ KEM and digital signature algorithms.

Although NIST has decided not to advance the NTRU a recent study [13] adapts it to IoT edge devices running Contiki-NG operating system. Performance of key pair generation, encryption, decryption, encapsulation, and decapsulation is measured in terms of time, stack usage, average power, and consumed energy. However, no information is given about the communication medium and protocol. In addition, digital signatures are not in the scope of the study at all.

Another closely related work is reported in [14], where the efficiency of PQ KEM algorithms are considered for securing IoT sensor devices. This work differs from ours as its main focus is on evaluating KEM schemes instead of signature algorithms, and uses a slightly different pool of KEM algorithms than we do. The authors are also concerned with the RAM and CPU usage of the devices themselves whereas we are concerned with the network traffic and overhead of the PQ algorithms.

III. Background

A. Transport Layer Security (TLS) Handshake

The TLS handshake is a series of messages between a server and a client machine that establish trust between the two parties and decide upon a shared secret key to encrypt their communications. Fig. 1 shows the steps of the TLS handshake process. TLS supports two approaches to establish this key: 1) *Diffie-Hellman (DH) key exchange*, and 2) *Key Encapsulation Mechanism (KEM)*. Note that in both cases, digital certificates for sharing public keys are exchanged as will be detailed in the next subsection.

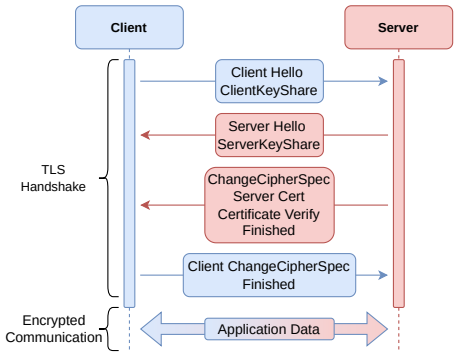

Fig. 1. TLS Handshake Messages

Security of the DH key exchange process relies on the discrete logarithm problem, in which two parties utilize randomly

generated private numbers to establish a shared secret key (the session key), whereas the KEM approach leverages public-key cryptography to exchange the session key between parties [7]. Fig. 2 lists the steps of Elliptic-Curve (EC) DH key exchange which is one of the variations of DH, and Fig. 3 shows the steps of KEM.

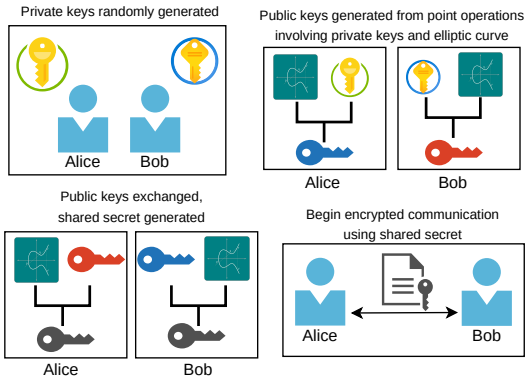

Fig. 2. Steps of Elliptic-Curve Diffie-Hellman Key Exchange

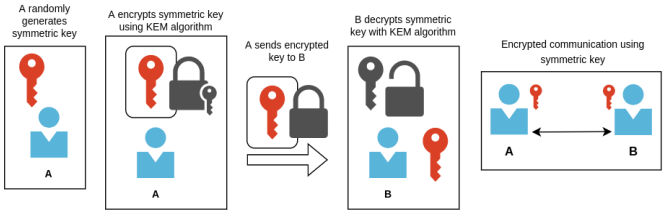

Fig. 3. Steps of Key Encapsulation Management (KEM)

B. Public Key Certificates and Mutual Authentication in TLS

A public key certificate is a digitally signed document that verifies to the recipient of a public key that the owner is legitimate. A certificate can be issued by a trusted organization that issues certificates, known as a Certificate Authority (CA). As can be seen in Fig. 1, in TLS only the server must send the client a certificate to verify its identity and public key (i.e., authenticate itself) to the client. However, optionally the client can send its certificate to the server as well in a process called *mutual authentication*.

C. Post-Quantum Algorithms

With the emergence of quantum computing, Shor's algorithm shows us that the commonly used discrete logarithm and integer factorization problems are relatively easy for a quantum computer to break. Hence, post-quantum cryptography aims to construct public-key cryptosystems that are believed to be secure even against quantum computers [7]. These approaches use a variety of mathematical functions, including lattice problems, zero-knowledge proofs, and learning with error problems.

The three signature scheme families (Dilithium, Falcon, and SPHINCS+) and one of the KEM scheme families (Kyber) that will be evaluated in our paper have been recently selected by NIST as its first four choices of post-quantum algorithms to be standardized in the future [15]. In addition to the new NIST standards, our assessment also includes the performance of additional KEM schemes: the NTRU_hps and Saber families of algorithms.

D. IP over BLE

There is no defined standard for IPv4 over BLE; the two technologies are normally incompatible, however, there have been some efforts such as RFC7668 [9] to establish a standard for running IP over BLE, though no official standard has been adopted as yet. 6LowPAN is utilized to reduce the header size of IPv6 packets.

IV. Approaches for TLS over BLE

A. Establishing IPv4 Over BLE Connections

While building a TLS over a WiFi-enabled connection is well-known and widely used, this is not the case for BLE since it is known for its deployment among two paired devices with its own BLE protocol stack. Therefore, the first challenge was to build an IP network by utilizing BLE as the link layer (layer 2) so that eventually this connection will get you to any machine in the world with an IP address. It is important to note that this will require assuming a gateway device (i.e., replacing the Access Point concept in WiFi) where the BLE device gets connected. As an example, a mobile phone can act as a device that will forward any BLE connection to itself to the rest of the Internet.

Since most of the existing efforts focused on IPv6 over BLE, we resort to a fairly simple method of running IPv4 over BLE by leveraging tools available on Linux. Linux uses the *Bluez* Bluetooth stack, which possesses some APIs allowing it to manipulate the traffic/operation with some flexibility. Using the tools through Bluez in conjunction with a virtual network interface created on both the client and server devices, we redirected IPv4 packets passed to the virtual Ethernet interface through a BLE physical interface.

In addition, we propose to host a DHCP server on the gateway device to issue IP addresses to connecting devices for the BLE interface, and this command includes the issuance of an IP address to the connecting IoT device. Note that a mobile phone acting as a gateway can also support DHCP services. We need this IP connectivity between the devices as TLS relies on underlying IP-based communication to complete the connection. Assigning IP addresses to the IoT device allows it to contact the gateway and send/receive data over the network using TLS.

After establishing an IP over the BLE network, the next step is using open-source quantum-resistant algorithms (e.g., Open Quantum Safe library of liboqs) to enable a TLS connection. Since TLS is also available through open source tools such as OpenSSL, a TLS handshake can be separately created for only assessment purposes (i.e., the machines do not transmit any application data).

The handshakes in TLS may use two different approaches, one using only the PQ signature schemes with ECDH key exchange, and the other using the PQ signature schemes in tandem with PQ KEM schemes.

B. First Approach - ECDH Based Key Exchange

The first approach involves using PQ-signed certificates and ECDH for key exchange. In ECDH, both parties randomly generate their own private key and use it for point operations on a specified elliptic curve to generate a public key. The public keys are shared between the two parties, who then conduct more point operations on the same curve in order to arrive at the same point, known as the shared secret, which is used to encrypt and decrypt communications between the two parties. The X25519 elliptic curve is used over the P256 curve in this approach because X25519 requires less processing power than P256 [16], making it useful to consider for the devices with limited processing resources such as IoT.

For our tests, the post-quantum signature algorithms are used to sign self-signed certificates (as opposed to RSA or ECDSA) verifying the identity of the server (and client if mutual authentication is required). The server machine sends its certificate to the client during the TLS handshake, after which the client needs to verify the PQ-signature of the certificate to validate the authenticity of the server. We assume that the PQ public key of the CA will be available at the client (i.e., as part of the browsers).

C. Second Approach - PQ Key Encapsulation Mechanism

The second approach also involves PQ self-signed certificates but uses PQ KEM schemes for key exchange. KEM involves one party randomly generating a secret key, then encrypting it with the KEM algorithm. The first party then sends the encrypted key to the second party, and the second party then uses the same KEM algorithm to decrypt it. Once the secret key has been shared with the second party, both parties can use it to securely encrypt their communications.

In the first approach, only the signature of the server's certificate uses PQ-cryptography, and the keys that encrypt the communications use the standard ECDH approach for key exchange, where both parties individually generate a shared secret key based on mathematical computations on an elliptic curve. The approach of combining post-quantum KEM and signature schemes instead uses key encapsulation, where both machines send the other a secret key encrypted by the destination machine's public key.

V. Performance Evaluation

A. Implementation and Experiment Setup

In order to evaluate the performance of post-quantum algorithms, we set up a testbed and performed various tests.

Hardware and Software: We used a Raspberry Pi 4 and a laptop to simulate a connection between IoT devices on a Bluetooth network by using an IP-over-BLE connection. The Raspberry Pi 4 used in this experiment has 4GB RAM, Bluetooth 5.0 and BLE capabilities, and a 1.8GHz ARM processor. The laptop used in this experiment runs Ubuntu 20.04 and has 8GB RAM, Bluetooth 5.0, and a 1.8GHz Intel i5 8^{th} gen. processor.

In order to use post-quantum algorithms over a TLS connection between a client and server machine, first *liboqs* must be downloaded and built from the official GitHub repository on both machines. To install *liboqs* on the laptop, the commands outlined in the README of the OpenSSL 1_1_1 fork [17] can be followed. However, the Raspberry Pi requires cross-compilation due to its ARM processor.

The Pi obtained an IP address over the network and used this address to identify itself and send/receive data over the network. OpenSSL was then used to establish a server and a client on this Bluetooth network, and to send a series of consecutive TLS handshake messages between the client (Pi) and the server (laptop). During the transmission of the data by s_time, the data packets containing the TLS handshake messages were recorded using Wireshark to determine the total wall-clock time per run, and then analyze to determine the most effective PQ algorithm for use in a Bluetooth-connected environment. Note that this setup can be reproduced on any machine that can use OpenSSL, including typical Windows, Mac, or Linux machines.

Fig. 4 illustrates the setup that we propose in simulating the IoT environment. In this setup, the Pi simulates an IoT device that connects to the laptop, which simulates a server machine. The connections are conducted at a set distance of 30 feet (10 meters) between the machines to better highlight the differences between Bluetooth and WiFi performance.

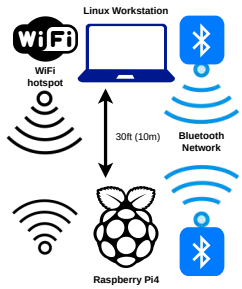

Fig. 4. Testbed Implementation.

IP Connections: Initially, we configured the laptop as a Network Access Point (NAP), and bind the interface to the Bluetooth radio. This is made possible through the use of an open-source Linux software package called *bluez-tools* [18]. Bluez-tools provides some additional scripts, commands, and tools which implement various aspects of the Bluez API, with the goal of simplifying working with Bluez - the official Linux Bluetooth protocol stack [19]. The Raspberry Pi is then paired with the laptop over Bluetooth to establish a BLE network, and IP connectivity is established.

The WiFi approach was implemented by using the built-in "WiFi hotspot" functionality of the laptop to enable a direct wireless connection between the Pi and laptop for a fair comparison with BLE. It followed the same steps as above for data collection.

Measuring the Connection Speed: To measure the connection speed of each algorithm, we set s_time to be run for 30 seconds per algorithm while the connection is recorded with the use of Wireshark. Since s_time displays user time instead of wall-clock time, the built-in timer in Wireshark is used to record the actual runtime of each s_time run. This time is then divided by the number of connections displayed by s_time, which provides us with the average time in seconds it takes each PQ algorithm to complete one connection.

B. Metrics and Baselines

We ran tests to analyze the connection speed of each post-quantum algorithm that was being considered. The main metric we are concerned about is how quickly the TLS handshakes can be conducted (i.e., TLS Handshake delay) using these algorithms over a BLE or WiFi network.

As a baseline for comparison to the PQ algorithms, we recorded data for connections using the current standards such as RSA and ECDSA. Also, we used WiFi as a benchmark to compare against since it offers a higher bandwidth.

C. PQ Signature Schemes with ECDH Performance Results

This section details the performances of post-quantum signature schemes when paired with ECDH Exchange along with their signature sizes. Table I lists the performance results for each PQ signature algorithm over BLE connection, with WiFi connection performance and the standard signature schemes RSA and ECDSA included as baselines for comparison.

TABLE I
TLS HANDSHAKE DELAY OF ECDH WITH DIFFERENT SIGNATURES (MS)

Signature Scheme	Signature Size	Bluetooth	WiFi
RSA	2048 bits	69.4	12.2
ECDSA	576 bits	59.9	8.7
Dilithium2	**19360 bits**	**123.3**	**10.6**
Dilithium3	26344 bits	150.3	12.3
Dilithium5	36760 bits	180.4	14.7
Falcon-512	**5328 bits**	**78.7**	**10.2**
Falcon-1024	10240 bits	104.1	12.1
SPHINCS+ Haraka	136704 bits	484.6	88.4
SPHINCS+ SHA	136704 bits	509.5	56.5
SPHINCS+ SHAKE	136704 bits	695.9	286.3

The results indicate that the Falcon-512 signature scheme is the fastest over a Bluetooth connection, with speeds closer to the baseline than any of the other chosen signature schemes. Over WiFi, however, Dilithium2 also shows comparable performance to the baseline results. It is likely that the performance of Dilithium2 decreased over BLE since the Dilithium family of signature schemes have a larger signature size than Falcon [17]. This would cause some extra delay in a lower-bandwidth environment like Bluetooth because the larger the signature size, the more time it requires to transmit the certificate. The SPHINCS+ signature schemes performed the worst over both BLE and WiFi due to delay incurred by their massive signature sizes.

An interesting observation here is among Dilithium2 and Falcon-512. We see that when the signature size is increasing, this has a major impact on lower bandwidth connections such as BLE. However, this is not the case with WiFi. Even though the signature size of Dilithium2 is almost 4 times more than Falcon-512, the delays for BLE and WiFi are still very close to each other (i.e., 10.6 vs 10.2 ms). This suggests that Falcon-512 requires more computation during the verification process. In other words, there are additional computational delays. Nevertheless, this is about the machine speed, not the connection. Therefore, signature size by itself should not be the sole indicator when selecting a PQ algorithm. Depending on the security level provided, higher bandwidth connections can afford bigger signature sizes. For BLE-like connections both the signature size and computation time are important.

TABLE II
TLS HANDSHAKE DELAY FOR SIGNATURE AND KEM SCHEMES (MSECS)

Signature	KEM Algorithm		
	Kyber512	Kyber768	Kyber1024
RSA-WF	12.6	12.9	14.1
RSA-BT	83.6	94.6	113.6
ECDSA-WF	9.4	10.0	11.7
ECDSA-BT	71.8	76.6	91.9
Dilithium2-WF	**11.4**	**11.6**	**12.1**
Dilithium2-BT	**135.4**	**157.4**	**167.8**
Dilithium3-WF	12.6	14.1	14.5
Dilithium3-BT	162.4	180.1	188.8
Dilithium5-WF	15.4	16.6	17.0
Dilithium5-BT	197.1	218.3	221.7
Falcon-512-WF	**10.7**	**11.2**	**12.0**
Falcon-512-BT	**92.9**	**100.5**	**116.8**
Falcon-1024-WF	**11.4**	**11.4**	**12.1**
Falcon-1024-BT	**119.4**	**127.5**	**133.0**
S+ Haraka-WF	89.7	90.6	91.6
S+ Haraka-BT	495.8	514.7	523.2
S+ SHA-WF	58.5	59.3	63.8
S+ SHA-BT	535.2	550.3	568.6
S+ SHAKE-WF	288.2	289.7	292.0
S+ SHAKE-BT	684.9	725.1	740.5

D. PQ Signature and KEM Schemes Performance Results

Our second experiment assessed the performances of PQ signature and KEM schemes, where a PQ signature included in a certificate sent in a TLS handshake that used PQ KEM for key exchange. Table II displays the average connection delays for combinations of PQ signature and KEM schemes over BLE (abbreviated BT), and the same over WiFi (abbreviated WF). SPHINCS+ is abbreviated as S+ in Table II for layout purposes.

First of all, it is important to note that due to the added complexity of the PQ key management schemes, all combinations showed slightly worse performance than using ECDH key exchange. The gap is higher with BLE. We can see that there is at least 10% delay increase which shows that for BLE like connections ECDH solutions could be preferred.

Among KEMs, we found that Falcon-512 was consistently the fastest signature scheme, followed by Falcon-1024 and Dilithium2. Across both WiFi and BLE, the combination with the fastest overall performance was Falcon-512 combined with Kyber512. Combinations including any of the SPHINCS+ signature schemes performed least favorably, with each consistently having a connection delay of over half a second over BLE and over two-tenths of a second over WiFi. The relatively massive delay seen with the SPHINCS+ family is

attributable to their very large signature sizes [17] causing lag in the limited-bandwidth environment of the BLE network, and to a less pronounced extent in the higher-bandwidth WiFi connections.

To summarize, in every case, BLE connections were much slower than WiFi connections but the most interesting point is that the Falcon family performs almost as well as RSA or ECDSA for both WiFi and BLE which makes it the top candidate for standardization for IoT applications.

E. Evaluation of Mutual Authentication

As it is possible for the client to send a certificate to the server as well (i.e., mutual authentication), we also performed some experiments to assess the overhead in this case compared to the single authentication approach. Since the logical outcome was that mutual authentication would increase the overhead further, we only selected the top candidates; Dilithium2 and Falcon-512 from the previous results to display this trend. The results are shown in Table II.

TABLE III
TLS DELAY WITH MUTUAL AUTHENTICATION (MSECS)

Signature	ECDH No KEM	Kyber512 KEM	Kyber768 KEM	Kyber1024 KEM
Falcon512-WF	30.6	31.1	32.2	33.2
Falcon512-BT	**187.2**	**192.2**	198.7	204.3
Dilithium2-WF	**16.9**	**17.2**	17.3	18.8
Dilithium2-BT	256.6	279.8	287.6	285.9

As can be logically assumed due to the extra certificate transmission and verification required, all cases of mutual authentication produced average results slower than the performances of single authentication in Table II. As an example, we will compare the differences in latency of Falcon-512. Over WiFi, Falcon-512 combined with Kyber512 had an average latency of 10.7ms with single authentication, but 31.1ms with mutual authentication (i.e., tripled). Over BLE, the difference is almost doubled; with average latency being 92.9ms using single authentication and 192.2ms with mutual authentication.

However, there is another interesting observation here that is consistent with our findings in Section V.D. The mutual authentication impacted Falcon512 much more than Diltihium2. If we look at Dilithim2 results with WiFi for Kyber512, the delay is 17.2ms compared to 11.4ms when we had single authentication. This means that the signature verification process in Falcon512 is much higher, and thus, it makes Dilithium2 a better alternative for WiFi if mutual authentication is to be applied.

VI. CONCLUSION

In this paper, we assessed the feasibility of various PQ algorithms for use in an IoT environment that uses wireless connections such as BLE and WiFi. The experiment results indicate that the signature schemes Falcon-512 and Falcon-1024 are generally the fastest in a BLE environment, regardless of being combined with ECDH or a PQ KEM scheme. The best overall performance over the BLE network was the combination of a Falcon-512 signature with Kyber512 for KEM. Surprisingly, Dilithium2 and Falcon-512 outperformed RSA over the WiFi connection, although this was not the case at all over Bluetooth. Dilithium2 and Falcon-512 proved to be the fastest signature schemes, and Kyber512 was the most efficient KEM scheme over WiFi.

ACKNOWLEDGMENT

This work is supported by the US NSF under the grant numbers CNS 2150248 and RINGS 2147196.

REFERENCES

[1] E. Knill, "Quantum computing," Nature, vol. 463, no. 7280, pp. 441–443, 2010.
[2] A. Ekert and R. Jozsa, "Quantum computation and shor's factoring algorithm," Reviews of Modern Physics, vol. 68, no. 3, p. 733, 1996.
[3] Y. I. Manin, "Classical computing, quantum computing, and shor's factoring algorithm," Asterisque-Societe Mathematique De France, vol. 266, pp. 375–404, 2000.
[4] A. Mandviwalla, K. Ohshiro, and B. Ji, "Implementing grover's algorithm on the ibm quantum computers," in 2018 IEEE international conference on big data (big data). IEEE, 2018, pp. 2531–2537.
[5] H. T. Larasati and H. Kim, "Quantum cryptanalysis landscape of shor's algorithm for elliptic curve discrete logarithm problem," in International Conference on Information Security Applications. Springer, 2021, pp. 91–104.
[6] F. de Lima Marquezino, R. Portugal, and C. Lavor, "Shor's algorithm for integer factorization," in A primer on quantum computing. Springer, 2019, pp. 57–77.
[7] D. Stebila and M. Mosca, "Post-quantum key exchange for the internet and the open quantum safe project," in International Conference on Selected Areas in Cryptography. Springer, 2016, pp. 14–37. [Online]. Available: https://www.mdpi.com/1424-8220/22/2/489/htm
[8] National Institute of Standards & Technology, "Post-quantum cryptography," Jul. 22, 2022. [Online]. Available: https://csrc.nist.gov/Projects/post-quantum-cryptography
[9] J. Nieminen, T. Savolainen, M. Isomaki, B. Patil, Z. Shelby, and C. Gomez, "Rfc 7668-ipv6 over bluetooth low energy," IETF: Fremont, CA, USA, 2015.
[10] R. Hernandez and Y. Hanna, "Quantum resistant cryptography over lora," 2021, unpublished.
[11] C.-C. Chung, C.-C. Pai, F.-S. Ching, C. Wang, and L.-J. Chen, "When post-quantum cryptography meets the internet of things: an empirical study," in Proceedings of the 20th Annual International Conference on Mobile Systems, Applications and Services, 2022, pp. 525–526.
[12] K. Bürstinghaus-Steinbach, C. Krauß, R. Niederhagen, and M. 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, 2020, pp. 841–852.
[13] J. Señor, J. Portilla, and G. Mujica, "Analysis of the ntru post-quantum cryptographic scheme in constrained iot edge devices," IEEE Internet of Things Journal, 2022.
[14] J.-A. Septien-Hernandez, M. Arellano-Vazquez, M. A. Contreras-Cruz, and J.-P. Ramirez-Paredes, "A comparative study of post-quantum cryptosystems for internet-of-things applications," Sensors, vol. 22, no. 2, 2022. [Online]. Available: https://www.mdpi.com/1424-8220/22/2/489
[15] National Institute of Standards & Technology, "Nist announces first four quantum-resistant cryptographic algorithms," Jul. 5, 2022. [Online]. Available: https://www.nist.gov/news-events/news/2022/07/nist-announces-first-four-quantum-resistant-cryptographic-algorithms
[16] D. J. Bernstein, "Curve25519: New diffie-hellman speed records," in Public Key Cryptography - PKC 2006, M. Yung, Y. Dodis, A. Kiayias, and T. Malkin, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 207–228.
[17] "Oqs-openssl_1_1_1," Feb 2022. [Online]. Available: https://github.com/open-quantum-safe/openssl/blob/OQS-OpenSSL_1_1_1-stable/README.md
[18] A. Orlenko, "Khvzak/bluez-tools: A set of tools to manage bluetooth devices for linux." [Online]. Available: https://github.com/khvzak/bluez-tools
[19] "Bluez about." [Online]. Available: http://www.bluez.org/about/