Journal of
Cybersecurity and Privacy

MDPI

Article

Post-Quantum Authentication in the MQTT Protocol

Juliet Samandari *,† and Clémentine Gritti †

Department of Computer Science and Software Engineering, University of Canterbury, Christchurch 8014, New Zealand
* Correspondence: juliet.samandari@pg.canterbury.ac.nz
† These authors contributed equally to this work.

Abstract: Message Queue Telemetry Transport (MQTT) is a common communication protocol used in the Internet of Things (IoT). MQTT is a simple, lightweight messaging protocol used to establish communication between multiple devices relying on the publish–subscribe model. However, the protocol does not provide authentication, and most proposals to incorporate it lose their lightweight feature and do not consider the future risk of quantum attacks. IoT devices are generally resource-constrained, and postquantum cryptography is often more computationally resource-intensive compared to current cryptographic standards, adding to the complexity of the transition. In this paper, we use the postquantum digital signature scheme CRYSTALS-Dilithium to provide authentication for MQTT and determine what the CPU, memory and disk usage are when doing so. We further investigate another possibility to provide authentication when using MQTT, namely a key encapsulation mechanism (KEM) trick proposed in 2020 for transport level security (TLS). Such a trick is claimed to save up to 90% in CPU cycles. We use the postquantum KEM scheme CRYSTALS-KYBER and compare the resulting CPU, memory and disk usages with traditional authentication. We found that the use of KEM for authentication resulted in a speed increase of 25 ms, a saving of 71%. There were some extra costs for memory but this is minimal enough to be acceptable for most IoT devices.

Keywords: Message Queue Telemetry Transport (MQTT); postquantum authentication; Internet of Things (IoT)

Citation: Samandari, J.; Gritti, C. Post-Quantum Authentication in the MQTT Protocol. *J. Cybersecur. Priv.* **2023**, *3*, 416–434. https://doi.org/10.3390/jcp3030021

Received: 13 May 2023
Revised: 15 June 2023
Accepted: 12 July 2023
Published: 31 July 2023

1. Introduction

The Internet currently uses different systems to provide secure communication. The transport layer security (TLS) and IP security (IPsec) protocols, along with the public key infrastructure (PKI) framework, are all commonly used communication procedures with security on the Internet. TLS is designed to block attempts to tamper with messages or listen in on what is being communicated [1]. IPsec is used in virtual private networks (VPNs) to authenticate and encrypt messages between two devices [2]. PKI ensures that public keys for devices are available and authenticated by using X.509 digital certificates with digital signatures [3]. Authentication is important, as online communication requires assurance that the communication is occurring between the expected parties and that the exchanged data are not being altered. Standards that can address this issue are the digital signature algorithm (DSA) based on the RSA (Rivest–Shamir–Adleman) cryptosystem or elliptic curves for authentication [4], and the advanced encryption standard (AES) for confidentiality [5].

Quantum computers are a new type of computer that uses qubits rather than bits. This means that the computer is no longer using just 0 or 1, but a superposition of both. Therefore, there are now infinite possible values for each qubit, as it can be any value from 0–1, inclusive. There are quantum algorithms that can be used by quantum computers to break our current security standards. The two main algorithms of note are Shor's algorithm [6], a quantum algorithm that can be used to find the prime factors of a value; and Grover's algorithm [7], a quantum algorithm that speeds up the search algorithm

from $O(N)$ to $O(\sqrt{(N)})$. According to Shor's algorithm, RSA and elliptic curve cryptography (ECC) will be broken [6], and with Grover's algorithm, AES will need double the key size [8], as we will not be able to rely on current standards for secure Internet communication once quantum computers are available [9]. This makes it necessary for new Internet communication standards to be created to ensure secure communication into the future. Classic computers (computers that rely on classical computing mechanisms for their operation) need new cryptosystems to be considered for Internet communication standards that remain secure against future attacks from quantum computers. These cryptosystems are called postquantum cryptosystems.

Physical quantum computers are yet to be commercially available. However, Microsoft has already created a cloud platform that provides quantum tools (https://medium.com/swlh/quantum-computing-is-now-publicly-available-df1abbc38578 (accessed on 7 February 2023)), Google has a 54-qubit machine called Weber (https://quantumai.google/hardware/datasheet/weber.pdf (accessed on 7 February 2023)) and IBM has a 127-qubit quantum computer called Eagle (https://immutabledistribution.com/this-insane-quantum-computer-is-ibms-last-chance/ (accessed on 7 February 2023)). Currently available quantum computers are unable to break current cryptosystems. However, quantum computers have been progressing greatly, and it is projected that they will become available in 20–30 years [10]. It is also expected that the number of organizations working in the area of quantum computing will nearly double in the coming years (https://www.techrepublic.com/article/6-experts-share-quantum-computing-predictions-for-2021/ (accessed on 7 February 2023)). It has been found that using an algorithm that simulates a quantum algorithm on a classic computer is up to 10 million times slower than when the algorithm is run on a quantum computer [11]. Exact differences in computing power are yet to be known, but it is expected that quantum computers will change computing as we know it at present.

The Internet of Things (IoT) is an incredibly versatile term and has many possible applications, including industrial and domestic implementations, security, logistics and transport. It is expected that the uses for IoT will increase, as will the number of connected IoT devices [12]. This makes it imperative that these new cryptographic standards be considered specifically for IoT devices. Due to the prevalence that IoT devices have, and are projected to continue having, it is imperative that there be security options when quantum computers become available [13]. It is especially crucial to be considering postquantum security options for IoT devices, because one of the major concerns for users of these devices is the possibility of security issues [12].

Message Queue Telemetry Transport (MQTT) (https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html (accessed on 26 January 2023)) is a lightweight application layer protocol commonly used for IoT communication. MQTT has no in-built security features, and usually runs over the TCP transport layer protocol. It is designed to be efficient and reliable, even if the network is unreliable. It is also easily scalable, so can be used for many connected devices (https://mqtt.org (accessed on 26 January 2023)). MQTT can be configured to run over the secure TLS protocol [1] to make the MQTT communication secure. However, this comes with the overheads associated with TLS, losing many of the advantages that MQTT has as a lightweight protocol.

1.1. MQTT

MQTT is a common communication protocol used in IoT. It is a simple, lightweight messaging protocol used to establish communication between multiple devices relying on the publish-subscribe model. The MQTT protocol is based on two types of parties: clients and brokers (sometimes referred to as servers). For our experiments, the IoT devices are clients and the gateway is the broker. All clients connect only to the broker. When first connecting, the client and broker perform a handshake, thus establishing their connection. Following this, there are two main types of messages exchanged: subscribe and publish. Clients can subscribe to certain topics (e.g., *temperature* or *humidity*), and the broker will store

a list of which clients subscribe to which topics. Clients can then send publish messages for a certain topic (e.g., *temperature*) and the broker will sort and forward the message to all clients subscribed to that topic. All clients can publish and subscribe to messages, although IoT devices often do not subscribe to many, if any, topics.

An example MQTT interaction can be seen in Figure 1. There, we have a system that has a gateway as a broker, and a temperature sensor and a mobile device as clients. The temperature sensor does not subscribe to any topics, and only publishes messages in the topic *temperature*. Therefore, there is a one-way connection from the temperature sensor to the gateway. The mobile device then receives updates about the temperature as it has subscribed to the *temperature* topic.

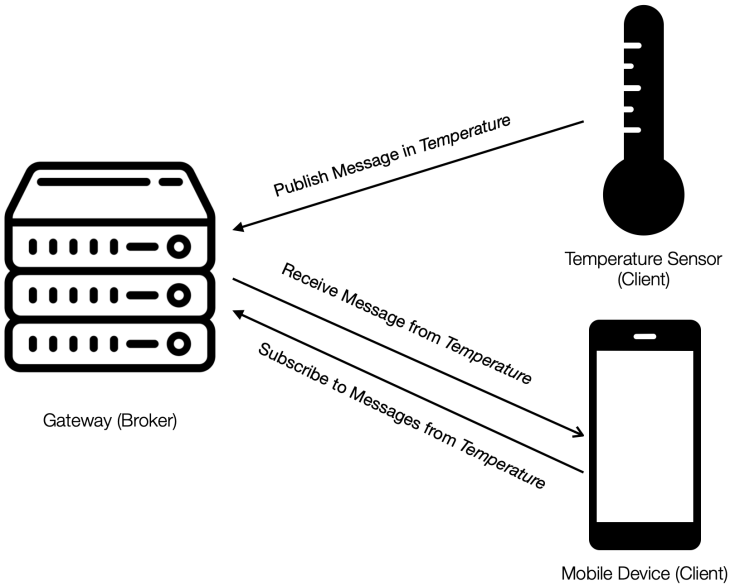

Figure 1. An example MQTT network.

MQTT uses TLS to provide authentication, so the handshake process is the same as the one used for TLS (https://www.ietf.org/rfc/rfc5246.txt (accessed on 1 February 2023)). This adds more overhead and reduces the lightweight advantage of MQTT.

1.2. Public-Key Cryptography

Public-key encryption and digital signatures are widely used, especially to provide assurance on the Internet [14]. Public-key cryptography can be used to provide confidentiality, integrity, authentication and nonrepudiation for individuals and organizations. PKI uses public-key cryptosystems to provide the basis of security for email, communication and transactions on the Internet [15].

1.2.1. Key Encapsulation Mechanisms

Public-key encryption requires the device to have two different keys: one public key to encrypt and one private (or secret) key to decrypt. These keys are used to facilitate secure communication between the devices by deciding on a shared symmetric key using a key agreement protocol. Postquantum key agreement works slightly differently, through a key encapsulation mechanism (KEM). With KEM, a key is generated and then encapsulated (similar to encryption) with the other devices's public key. The encapsulated key is then sent to the other party, which decapsulates the key (similar to decryption) [16]. This results in a shared secret key between the two devices.

1.2.2. Digital Signatures

A digital signature is used to verify the sender's identity and to check the integrity of a received message [17]. Digital signatures require three actions to be carried out: to

generate a key for signing, to sign a message using a secret key and to verify if a received signature is valid using a public key [18,19].

1.3. NIST Postquantum Standardization

The National Institute of Standards and Technology (NIST) ispart of the U.S. Department of Commerce. NIST is responsible for providing standards and support for technological advancements in the United States. Due to the international nature of technology, NIST standards are used around the world. In 2016, NIST began a competition for new, quantum-resistant (postquantum) cryptographic standards to be proposed (https://csrc.nist.gov/projects/post-quantum-cryptography (accessed on 7 February 2023)). The areas identified as needing new standards are encryption, key establishment, and digital signatures, as these are the areas that will no longer be secure in their current form with the use of quantum computers (https://www.nist.gov/news-events/news/2016/12/nist-asks-public-help-future-proof-electronic-information (accessed on 7 February 2023)).

Table 1 shows the five security levels that NIST has used as benchmarks for the proposed postquantum cryptographic schemes (https://csrc.nist.gov/Projects/Post-Quantum-Cryptography/Post-Quantum-Cryptography-Standardization/Evaluation-Criteria/Security-(Evaluation-Criteria) (accessed on 7 February 23)). Our implementations consider security levels 2, 3 and 5, corresponding to being as difficult to break as a collision search on a 256-bit hash function, a key search on a block cipher with a 192-bit key and a key search on a block cipher with a 256-bit key, respectively.

Table 1. NIST postquantum security levels.

Security Level	Classic Equivalent
1	AES-128 (as difficult to break as a key search on a block cipher with a 128-bit key)
2	SHA256 (as difficult to break as a collision search on a 256-bit hash function)
3	AES-192 (as difficult to break as a key search on a block cipher with a 192-bit key)
4	SHA384 (as difficult to break as a collision search on a 384-bit hash function)
5	AES256 (as difficult to break as a key search on a block cipher with a 256-bit key)

NIST Lightweight Protocol Standardization

In February 2023, NIST announced their selected suite of lightweight cryptography algorithms to be used for IoT devices. The selected suite is called ASCON [20]. This competition was first proposed in 2019, and of the 57 candidates, only one was selected for standardization. One of the variants in the ASCON suite may have some level of protection against quantum attacks. However, this was not one of the aspects NIST took into account in the selection of this cryptographic suite [21]. The main focus was to find an algorithm that would work for most resource-constrained devices, as the implementation is very compact.

2. Problem Statement

MQTT is a popular communication protocol for use by IoT devices, and is very useful in this context. However, MQTT was originally designed with no security, and the current options for security will add to the required communication costs in a way that may not be practical for resource-constrained devices. This is likely to only be exacerbated by the use of postquantum algorithms, as they can often require more processing power. While there have been prior works that explore postquantum cryptographic algorithms in the context of MQTT, these works have been relatively limited. Additionally, the work that has been conducted on postquantum MQTT has often not taken authentication into account. This highlights three major issues. The first is that MQTT does not itself provide authenticated communication. This means that it is necessary to use an additional function to authenticate the communication channel. The second is that the use of an additional mechanism for

authentication adds to the computation costs, which for constrained IoT devices can result in significant delays. Finally, it is necessary to find an option for authentication that has feasible computation expectations for IoT devices and also does not result in an increased number of round trips needed to establish the authenticated connection. Throughout our paper, we will address these issues by looking at providing authentication, the related costs and a possible way to reduce these costs through the use of a KEM.

2.1. Idea

Current methods to provide authentication in other protocols, such as TLS (https://www.ietf.org/rfc/rfc5246.txt (accessed on 26 January 2023)), use digital signatures, and therefore, this would be a logical way to provide authentication in the MQTT protocol.

Digital signatures are especially important for IoT networks to verify that all devices are legitimate users and to prevent the access of intruders to the network, as well as to prevent them from making undetected changes to communications between IoT devices that are part of this network. Since IoT devices are focused on information exchange, it is important to not only provide proper security for the communication in the form of encryption, but also to prevent illegitimate users from gaining access to the communication channels through the use of authentication that is performed using digital signatures [22].

NIST is currently in the process of defining a set of standard postquantum public key cryptographic algorithms (https://csrc.nist.gov/projects/post-quantum-cryptography/post-quantum-cryptography-standardization (accessed on 26 January 2023)), including postquantum digital signature algorithms. Therefore, following a similar handshake process to TLS, postquantum digital signatures could be used to provide authentication in MQTT.

However, as described in [23], we can expect significant overheads in terms of bandwidth, computational time and storage requirements with digital signatures. In [23], it is proposed that some efficiency gains can be made by authenticating using KEMs instead of signatures. KEMs are generally used to provide confidentiality through asymmetric encryption. This means that a public encryption key is available, which anyone can use to encrypt. However, only the owner of the public key has access to the private decryption key to read the encrypted message. The proposal is for TLS postquantum authentication, and promises less than 10% of the computational time for KEM-based authentication compared to a digital signature scheme.

2.2. Digital Signatures Used for Authentication

The traditional MQTT interaction over a public MQTT channel, with authentication provided by digital signatures, can be seen in Figure 2. The client sends a `ClientHello` message to the broker with a random nonce, r_c, for freshness. The broker responds with a `BrokerHello` message containing another random nonce, r_b, and then sends its public key signed by an authority, $\texttt{cert}[pk_s]$, verifying that the public key does indeed belong to the expected party. After receiving the signed public key, the client verifies the signature and then encapsulates a premaster secret, pms, using the public key. This encapsulation, ct_s, is then sent to the broker.

When the broker receives ct_s, it decapsulates ct_s with its secret key, sk_s. Both parties use the premaster secret, pms, to generate further keys that can later be used for symmetric encryption to provide confidentiality. Both parties then verify that the other party has indeed received all the messages that were sent to it, and that they have generated the same keys by sending hash message authentication codes (HMACs) back and forth in the `ClientFinished` and `BrokerFinished` messages. These messages allow both parties to check that the other has been able to calculate the same hash, based on the contents of all previous messages and the final generated key. This completes the authentication of the broker, meaning the clients can be sure that they are interacting with the intended broker. The client will then send the packets required to complete the MQTT handshake, and the protocol will continue as normal.

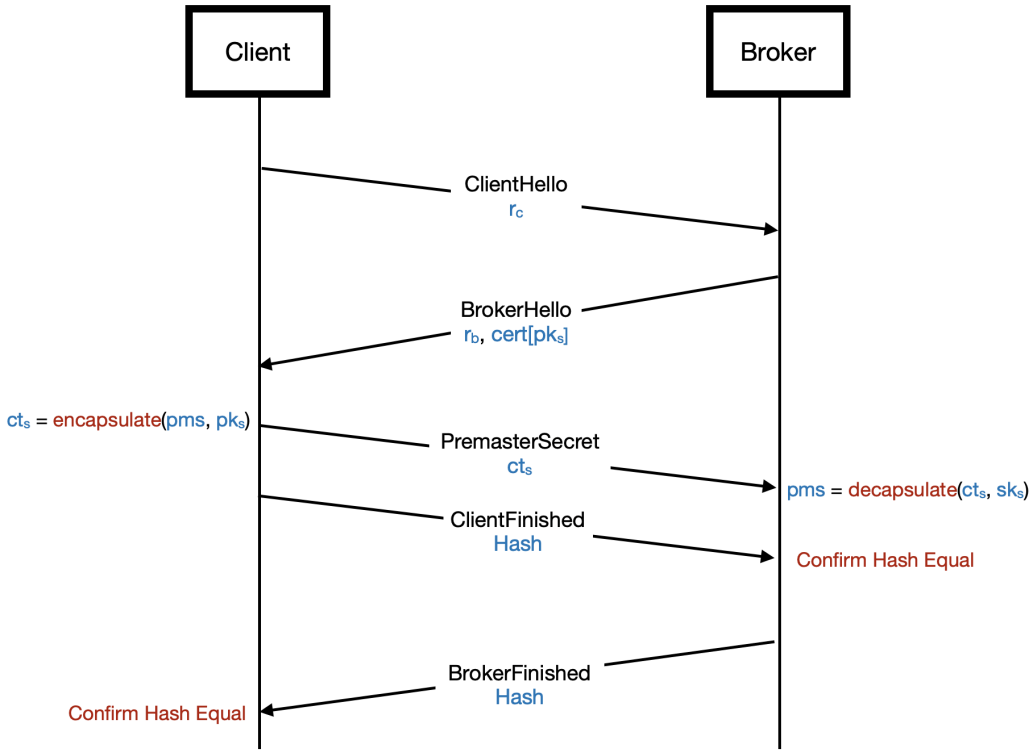

Figure 2. TLS handshake for MQTT authenticated using digital signatures.

2.3. KEM Used for Authentication

Our proposal for using KEM for authentication in MQTT is based on the proposed change to TLS found in [23]. KEM was proposed as an option to provide faster authentication than with postquantum digital signatures. The authors claim that there is a reduction of 90% in CPU cycles when KEMs are used.

The interaction for the handshake when authentication is provided by a KEM rather than a digital signature is slightly different and is outlined in Figure 3. This interaction is over a public MQTT channel. The client first generates an ephemeral secret and public key pair sk_e, pk_e. The client then sends a `ClientHello` message with a random nonce, r_c, and their public key. The broker then encapsulates an ephemeral shared secret, ss_e, using the public key and sends it to the client along with a random nonce, r_b, and their own public key, pk_s. The broker's public key is signed by an authority to verify that it does indeed belong to the intended broker (this signature cannot be avoided). The client verifies the signature and encapsulates a static shared secret, ss_s, with the broker's public key, and sends this to the broker.

The broker decapsulates the shared secret using their secret key, sk_s, and then both the client and broker generate several keys based on the shared secret. They then exchange HMACs of the previously exchanged messages and the final key to ensure that both parties have received the same messages and have calculated the same keys. At this point, the client has authenticated the broker and the client can be sure they are connecting to the broker they intend to. The rest of the MQTT protocol can continue as normal.

2.4. Security Analysis

As secure MQTT runs over TLS 1.3, the security analysis carried out in [23] remains relevant to our proposal of KEM-authenticated MQTT. In their paper, they outline the main security goal they focus on to be that the keys used at every stage of the KEMTLS (KEM-authenticated TLS) protocol to be indistinguishable from a random key. This should be the case even if the adversary has extra information, such as that it can view and control

all communication or know the long-term keys of devices not currently involved in the interaction. They also show that forward secrecy is maintained. The protocol uses a mix of implicit and explicit authentication before finishing the final stage with full explicit authentication. The amount of forward secrecy therefore differs depending on the stage of the user. For instance, the first two stages only have level 1 weak forward secrecy; this means that even if the long-term key is known, the secret key cannot be obtained. However, these keys have not been authenticated, and will not be until the full handshake is completed.

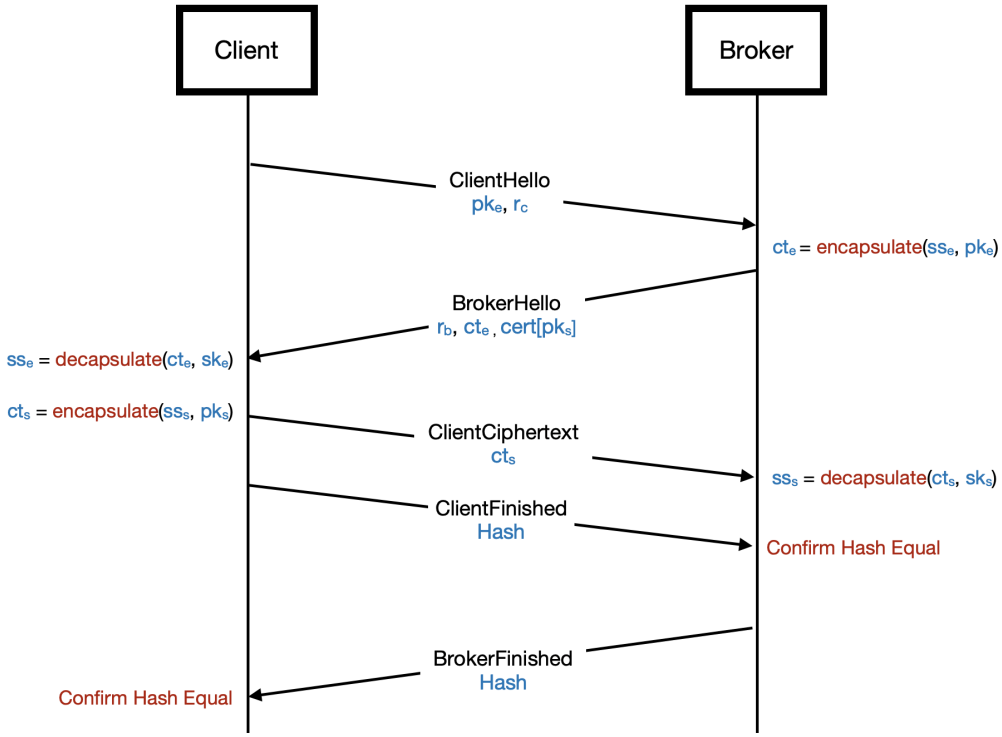

Figure 3. TLS handshake for MQTT authenticated using KEMs.

2.5. Contributions

This paper outlines how existing postquantum cryptography fits within the existing communication protocol, MQTT, to integrate authentication.

- We discuss the use of MQTT for IoT and provide background on its use.
- We discuss postquantum cryptosystems and test MQTT with the postquantum KEM and digital signature algorithms CRYSTALS-Kyber and CRYSTALS-Dilithium, respectively.
- We discuss the opportunity to use KEMs for authentication rather than digital signatures.
- We create a skeleton implementation of MQTT authenticated with the KEM CRYSTALS-Kyber to act as a proof of concept for the feasibility of the use of a KEM for authentication in MQTT.
- We implement a comparison of unauthenticated MQTT, MQTT authenticated with digital signatures (CRYSTALS-Dilithium) and authenticated with KEM (CRYSTALS-Kyber).
- We then compare and discuss our results in the context of the use of postquantum authentication in MQTT for IoT devices in the future.

3. Results

3.1. Results for Digital Signature Authentication

Results for digital signature authentication are all averages of the 60 repetitions carried out in the experiment.

3.1.1. CPU Usage

CPU usage is given as a percentage of the overall CPU available. In Figure 4, it can be seen that apart from the broker at dilithium3, the CPU usage either remained stable (for the subscriber with dilithium2, dilithium3 and dilithium5) or decreased (for all other instances) as the available memory decreased. However, the change in CPU usage was only ever at most a 0.1% difference.

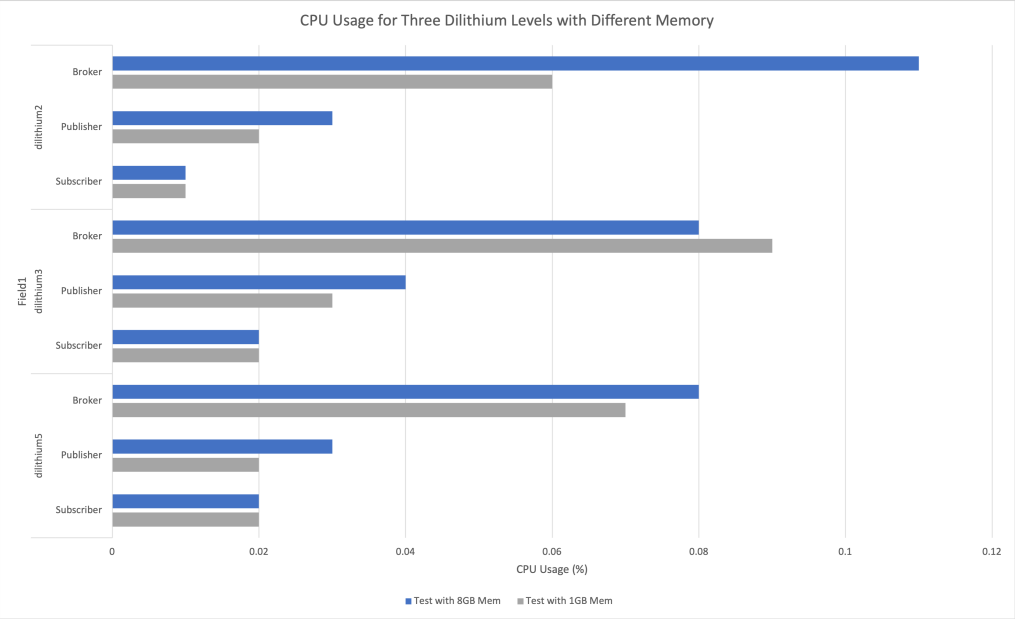

Figure 4. Percentage CPU usage When running MQTT.

It can also be seen that the CPU usage is consistently greater for the broker. The CPU usage for the publisher and subscriber is similar, but with the publisher's usage being slightly more than the subscriber's in many cases.

3.1.2. Memory Usage

Memory usage, as displayed in Figure 5, is seen to have a dramatic increase for the broker when running all three levels of the digital signature when there is only 1 GB of memory and 1 CPU core available. There is a jump from approximately 2 MB to about 7 MB, 12 MB and 10 MB for dilithium2, dilithium3 and dilithium5, respectively. The difference for other devices and security levels is much more subtle. Not considering the aforementioned circumstances, there was no increase in memory for the subscriber with dilithium2, but the Publisher used 1.84 MB for dilithium2 with the restricted test. The increase was between 0.46 MB and 0.48 MB for dilithium3. For dilithium5, there was an increase of 0.12 MB for the subscriber but a decrease in memory usage by 0.01 MB for the publisher.

Memory usage increases slightly as the security level increases when there is 8 GB of memory available. The overall change from dilithium2 to dilithium5 is less than 0.23 MB for all three devices in the interaction. The greatest difference is seen for the publisher, with 0.23 MB (0.4 MB from dilithium2 to dilithium3 and 0.19 MB from dilithium3 to dilithium5); then the subscriber, with 0.1 MB (0.4 MB from dilithium2 to dilithium3 and 0.06 MB from dilithium3 to dilithium5); and the least difference is seen for the Broker, with 0.08 MB (0.3 MB from dilithium2 to dilithium3 and 0.05 MB from dilithium3 to dilithium5).

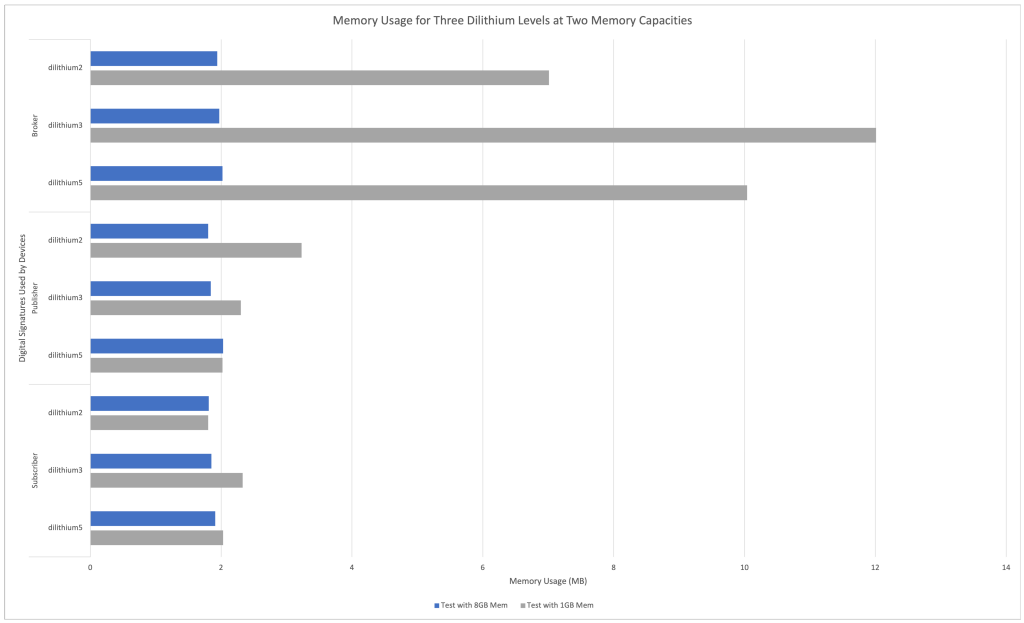

Figure 5. Memory usage when running MQTT

The memory usage sometimes increases with the security level when there is 1 GB of memory available. However, there were a few times when it decreased. For instance, the memory usage decreased for the publisher as the security level increased (3.23 MB for dilithium2, 2.3 MB for dilithium3 and 2.02 for dilithium5).

3.1.3. Disk Usage

Nothing is read from the disk when the available memory is 8 GB for any of the security levels of CRYSTALS-Dilithium. However, when there is 1 GB of available memory, the MQTT network begins to read from the disk, as seen in Figure 6. It can also be seen that the broker requires a lot more disk usage than the subscriber and publisher. The subscriber also seems to require more disk usage than the publisher.

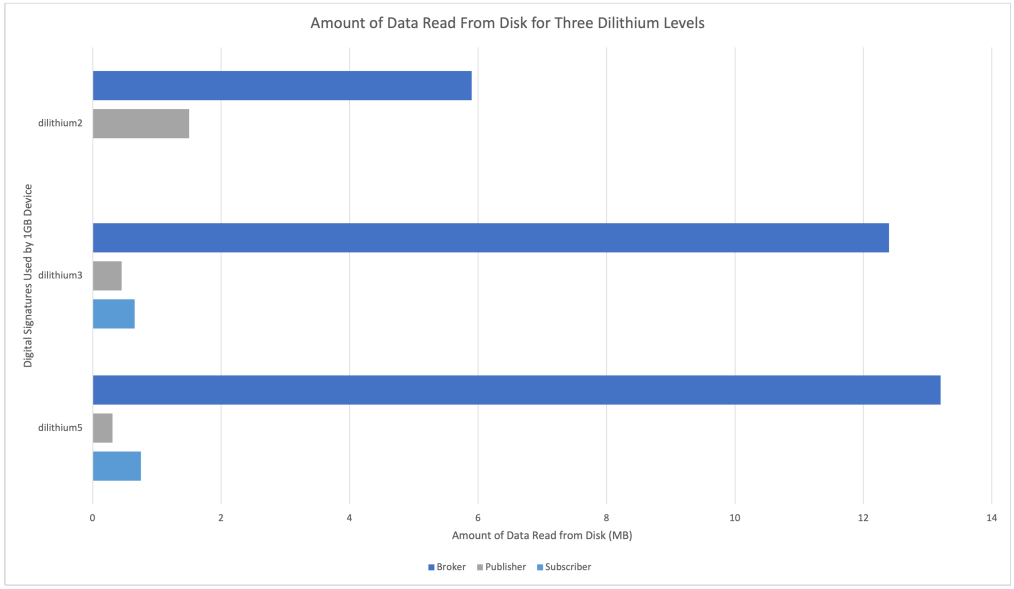

Figure 6. Disk usage when running MQTT on device with 1 GB of memory.

The amount read increases as the security level increases for all devices except the Publisher, where a slight decrease of 144 KB is seen. For dilithium2, only the broker and

subscriber read from the disk, with 5.9 MB and 1.5 MB read, respectively. For dilithium3, the broker reads 12.4 MB from the disk, while the subscriber and Publisher read 655 KB and 451 KB, respectively. The broker, subscriber and publisher read 13.2 MB, 750 KB and 307 KB from the disk, respectively, for dilithium5.

The amount written to the disk was consistent according to the security level. For dilithium2 and dilithium3, the broker, subscriber and publisher wrote 57.3 KB, 45.1 KB and 45.1 KB to the disk, respectively. For dilithium5, the broker, subscriber and publisher wrote 65.5 KB, 53.2 KB and 53.2 KB to the disk, respectively. There was no change seen at all with the reduction of available memory to 1 GB.

3.1.4. Time Taken

The time taken that is recorded is the addition of the user and sys times. This gives the CPU time taken to send a publish message to the broker. As can be seen in Figure 7 it took longer to send a message as the security level increased. It also took longer to send a publish message when there was only 1 GB of available memory rather than 8 GB of memory.

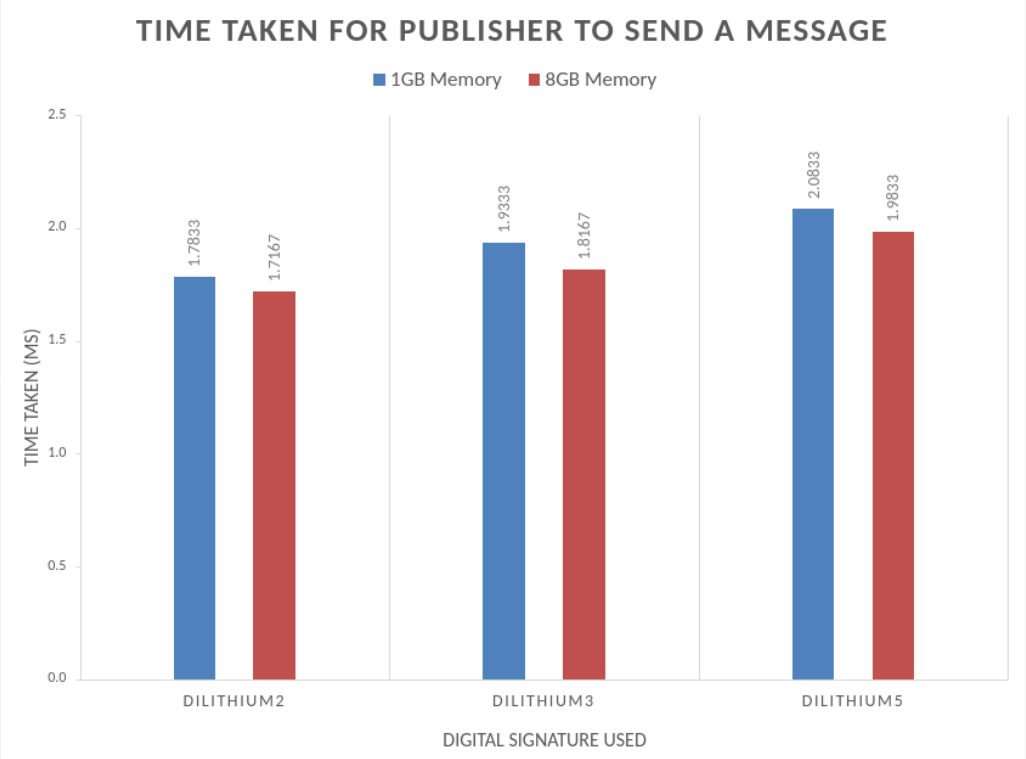

Figure 7. Time taken to send an MQTT publish message.

3.1.5. Network Communication

The amount of data sent over the network was not seen to change as the amount of available memory was reduced. Network output was consistently more for the broker at all security levels and with reduced memory, whereas network input was greater than output for the subscriber and publisher. It could also be seen that network input was relatively similar for all device types, from around 20 to 35 KB.

3.2. Results for KEM Authentication

Experiments were carried out 100 times, and the results seen below are the average values.

3.2.1. Storage Used

The size of public and secret keys that must be stored on the client were considered in order to give a general indication of the necessary storage. The storage on the broker

side was not deemed significant. The broker will usually be a machine that is relatively powerful and has ample storage for a few kilobytes of public and secret keys. The signature-based scheme was determined to require 11 KB of memory compared to 17 KB for the KEM-based scheme.

3.2.2. Bandwidth Used

To measure the memory bandwidth required by the signature-based and KEM-based MQTT protocols, the number and size of packets sent back and forth were monitored by writing the name and size of each packet sent and received on the client to a log file. The number of packets is the key factor when comparing bandwidth required by protocols, as more packets means more overhead at each lower layer in the network architecture. However, one protocol may have an advantage if the size of its packets is smaller. As is evident in Figures 2 and 3, both signature-based and KEM-based MQTT protocols send the same number of packets. Since both protocols have a similar number of packets, this means they will require similar amounts of bandwidth. Therefore, any advantage seen will be due to the packet size.

As shown in Figure 8, the KEM-based MQTT protocol has the largest total number of bytes to send at 13,524 B, compared to 7,732 B in the signature-based MQTT protocol. This is due to the shared secret ss_e being sent both ways in the handshake of the KEM-based protocol.

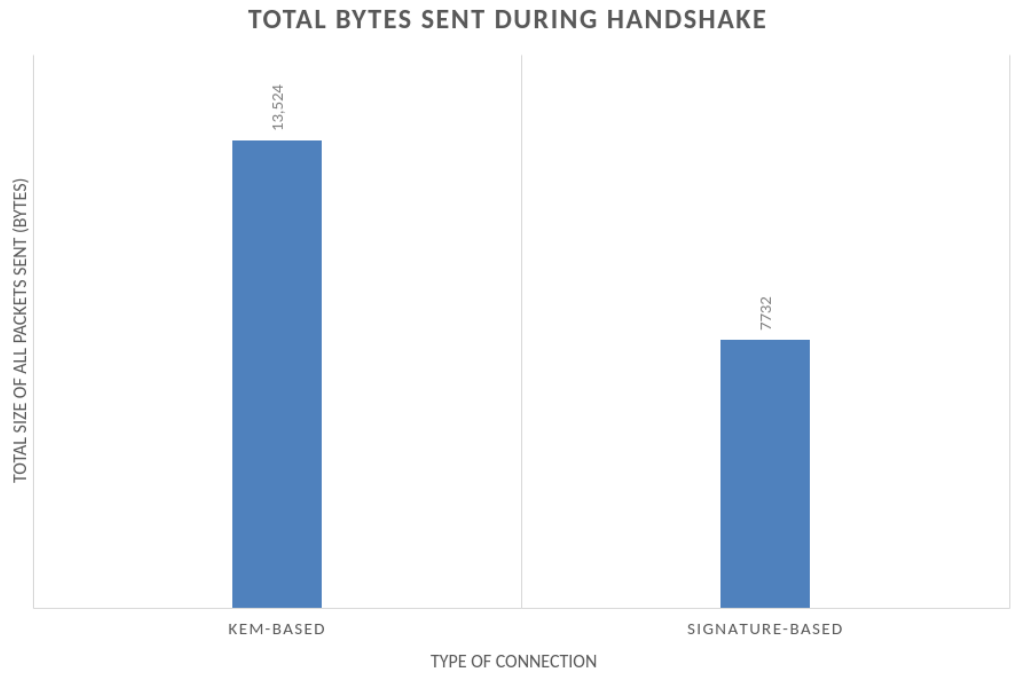

Figure 8. Total bytes sent during connection for KEM-based and signature-based protocols.

The KEM-based protocol also has the largest individual packet to send, at 3619 B for the `BrokerHello` message, compared to 2883 B for the `BrokerHello` message in the signature-based protocol. Given that the number of packets sent is equal, the bandwidth advantage of the signature-based version is minimal, with only a difference of approximately 1.5 KB.

3.2.3. Time Taken

To measure connection speed, the time elapsed for complete authentication was recorded for 100 handshakes in each protocol. The Python time library was used to write the difference between the clock time immediately before initiating the handshake and immediately after it was successfully completed. Results can be seen in Figure 9.

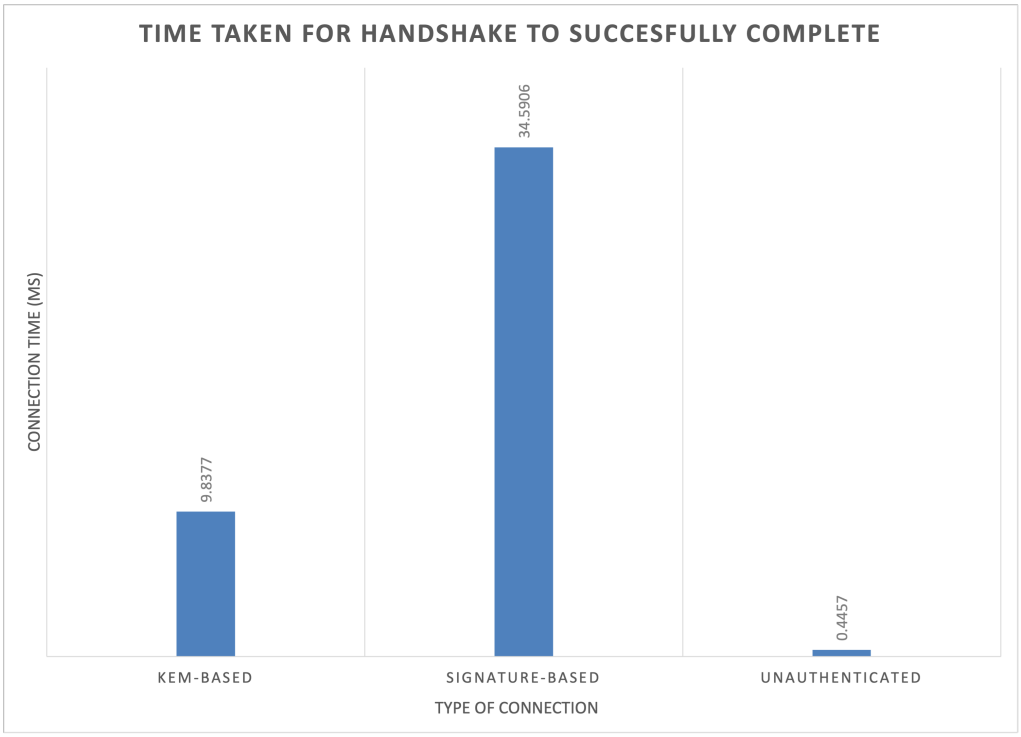

Figure 9. Average authentication time in KEM-based and signature-based protocols.

On average, the unauthenticated handshake took 0.4 ms, the signature-based authenticated handshake took 35 ms, and the KEM-based authenticated handshake took 10 ms. The KEM-based protocol was 71% faster than the signature-based protocol, but was still 22 times slower than the original MQTT protocol (i.e., without authentication).

4. Materials and Methods

Two sets of experiments were run: first, testing the use of digital signatures to authenticate MQTT, then testing the use of KEM to authenticate MQTT. The CRYSTALS library of postquantum cryptographic functions was chosen to be used for our experiments as CRYSTALS-Kyber is the only KEM selected to be standardized so far by NIST, and CRYSTALS-Dilithium has been found to be more efficient to run for all steps of the digital signature than the alternatives FALCON and SPHINCS+.

Our experiments are intended to give usage information about a generic MQTT interaction.

4.1. Implementation of MQTT Authenticated with Digital Signatures

Experiments were run on Docker (https://www.docker.com/ (accessed on 2 February 2023)), a container management software. Results were gathered using Docker's Research Usage extension (https://www.docker.com/blog/how-to-monitor-container-memory-and-cpu-usage-in-docker-desktop/ (accessed on 2 February 2023)). The Docker environment has 4 CPU Cores and 8 GB of memory allocated for its use. Docker was run on an Apple Computer with a 2.9 GHz Quad-Core Intel Core i7 with 16 GB RAM.

The interaction setup on Docker closely matches what is seen in Figure 1, with the difference that TLS is used to secure the MQTT connection; this is currently the only option for secure MQTT. There is a broker (matching the gateway), a publisher (matching the temperature sensor) and a subscriber (matching the mobile device). The network is set up and the publisher publishes 60 updates to the broker. The broker then forwards on these updates to the subscriber. The CPU, memory and disk usages, time taken and data sent over the network are monitored. This network was set up three times with the CRYSTALS-Dilithium digital signature scheme at security levels 2, 3 and 5 (referred to as dilithium2, dilithium3 and dilithium5, respectively, for brevity). This experiment was

repeated with limited Docker resources (memory limited to 1 GB, 1 CPU core, and swap memory limited to 512 MB) to view what effect the constraint would have on the function of the MQTT network.

The Docker image used for the experiments can be found at the demos fork of open-quantum-safe (https://github.com/anhu/oqs-demos/tree/wolfmqtt_compat/mosquitto (accessed on 2 February 2023)) where it builds Mosquitto (https://mosquitto.org/ (accessed on 2 February 2023)), an open source MQTT broker. Docker was chosen because our focus is on the postquantum scheme and its incorporation in MQTT, rather than on the considerations required when there are issues with the network connection.

We decided to use CRYSTALS-Dilithium as the digital signature algorithm of choice because NIST has concluded that CRYSTALS-Dilithium is well suited for most applications [24]. Additionally, Falcon, another to-be-standardized postquantum digital signature algorithm, uses floating point arithmetic, which is undesirable for IoT devices due to the greater computational costs required. Indeed, floating point arithmetic on more constrained devices is implemented in software, resulting in significantly slower calculations. There is also potential for side-channel attacks on Falcon as a result [25].

4.2. Implementation of MQTT Authenticated with KEM

We created skeleton implementations of MQTT with signature-based and KEM-based authentication using Python. Python implementations of postquantum algorithms were imported from the pqcrypto library (https://pypi.org/project/pqcrypto/ (accessed on 2 February 2023)). The skeleton implementations include separate packages for both the client and the broker. The client can act as both a publisher and subscriber. Only the part of the MQTT protocol required to demonstrate the postquantum authentication process was implemented, namely the handshake process. The implemented parts of the MQTT protocol were programmed exactly as per the documentation of MQTT, except the additional options, such as higher quality of service. The choice to use a minimal implementation of MQTT was made to focus on the objective of determining the impact of implementing postquantum authentication in MQTT. We used Kyber-512 (security level 1) and dilithium2 (security level 2) for our experiments. We used two virtual machines for the client and broker. These machines both had 1 GB of RAM and 8 GB of storage. The code used can be found in this GitHub repository (https://github.com/raven-townsend-nz/quantum-safe-mqtt (accessed on 9 May 2023)). As this implementation is similar to a proof of concept and not a full implementation, security considerations such as side-channel attacks, which are implementation specific, have not been considered.

5. Discussion

Some slight increases in CPU usage could be seen as available memory decreases when using digital signatures for authentication. However, the increases were too slight and inconsistent to result in any noticeable impact in CPU usage. The CPU usage for the broker was consistently greater than that for the publisher and subscriber. However, the broker can usually be expected to be the less constrained device (i.e., a gateway) compared to the publisher and subscriber.

There was an increase in memory usage for the broker when running dilithium3 and dilithium5 when there was only 1 GB of available memory. This might be linked to the need to read from the disk, something that did not occur when there were 8 GB of available memory. There was also a large amount of disk usage seen for the Broker when only 1 GB of memory was available.

The value written to the disk is likely consistent, as the size of the generated keys and certificates would be practically the same for all instances of a given security level. There was no change with the reduced available memory, which is as expected, since the size of the keys and certificates should be consistent.

Network output was more for the broker, which is likely because the broker is the gateway and responsible for communicating with both the subscriber and publisher, whilst the subscriber and publisher only communicate with the broker.

The time increase observed when the available memory is reduced is in the order of microseconds. This may be cause for concern when considering the use of authentication in MQTT for devices that are even more constrained, with memory in the order of megabytes or even kilobytes, as the increase in time could grow further.

Signature-based authentication in MQTT has a slight advantage in terms of storage, according to the metric we have considered, compared to KEM-based authentication. However, cases where a device can handle 11 KB but not 17 KB are unlikely, so this advantage is minimal.

Signature-based authentication in MQTT has a slight advantage in terms of bandwidth compared to KEM-based authentication. However, the bandwidth results are both in the same order of magnitude for both protocols, and it is therefore unlikely that there will be a case where the signature-based version is acceptable but the KEM-based one is not.

The KEM-based MQTT version has a significant advantage in connection speed. It was, on average, 25 ms faster than the signature-based version, which is a difference of 71%. In [23], a 90% reduction in CPU cycles was seen, which aligns with our results. A full 90% reduction was not expected, as the connection time we measured included the overhead of packet transmission, as well as the actual CPU processing time. This speed increase using KEM would allow subscribers and publishers to connect to and authenticate the broker relatively quickly compared to a traditional signature-based authentication approach.

Overall, KEM-based authentication has slight disadvantages in terms of bandwidth. However, it has a significant advantage in terms of connection speed. This suggests that in an IoT context, the KEM-based version may be the most suitable, particularly as one of the key focuses of MQTT is to be a fast communication protocol in IoT.

The extra costs of memory bandwidth and connection speed from adding authentication are low enough to be acceptable in most situations, particularly as CPUs become more powerful and efficient in IoT. However, there may be cases where these additional resources are not acceptable, especially when the IoT devices are quite resource-constrained.

6. Related Work

There have been several proposals for adding security to the MQTT scheme. It was suggested in [26] that implementing MQTT over TLS may be practical in most cases, as the increased capacity of IoT devices will make the additional overhead no longer a concern. In [27,28], secure versions of the MQTT protocol were designed using attribute-based encryption (ABE) [29]. In [27], the authors particularly focused on efficiency, meaning the computational power required for their proposal was less than in [28]. In [30], security architectures and procedures were proposed to provide authentication, integrity and confidentiality for MQTT-SN, a version of MQTT designed to run over the User Datagram Protocol (UDP). Similarly, in [31], the authors tried to address the issue of increased bandwidth by proposing to secure MQTT-SN over UDP rather than standard MQTT over TCP. UDP has smaller packet headers than TCP, so implementing security over MQTT-SN results in smaller packet sizes than implementing security on MQTT. However, in both papers, the issue of postquantum attacks was not addressed, as the authors suggested classical cryptography algorithms such as elliptic durve Diffie–Hellman [32], DTLS or LBlock, a lightweight block cipher [33]. In both papers, the authors concluded that the proposed architectures could provide these security outcomes with reasonable sacrifices to performance.

Postquantum capability has been added to wolfMQTT, a product of wolfSSL. wolfSSL is an open-source secure sockets layer (SSL) option that is lightweight and popular for use by IoT devices (https://www.wolfssl.com/about/ (accessed on 2 February 2023)). wolfMQTT now provides support for CRYSTALS-Kyber and Falcon at security level 1 (https://www.wolfssl.com/wolfmqtt-post-quantum-kyber-falcon/ (accessed on 2 February 2023)).

In [34], postquantum MQTT is implemented in an IoT environment and compared with classic algorithms. It is unclear at which security level the authors used these algorithms in, but they found that many postquantum cryptographic algorithms had a manageable increase in latency and overhead. In [35], an IoT network is implemented with MQTT, but only KEMs are tested, so this work does not include authentication. The authors found that postquantum cryptosystems performed similarly to their elliptic curve counterparts.

All the solutions described above provide security through tested methods, meaning that they do not have any known vulnerabilities. However, they have two key problems. Firstly, they increase the computational power and bandwidth required by IoT devices to run the secured protocol. Secondly, MQTT with classic cryptosystems will not be secure against quantum computer attacks once quantum computers become widely available. Also, the work that has been carried out for postquantum MQTT has been limited and has not necessarily taken into account authentication.

Other Postquantum Testing

Work has been more generally conducted on different lattice-based problems. In [36], they find that the ring-learning with errors (R-LWE) problem is the most practical and efficient. They then go on to propose R-BinLWE, where a binary distribution is used. They propose a novel multiplication technique that reduces area usage by 57.8% and power usage by 48.42% on the device. In [37] they implement a R-LWE encryption scheme on the ARM Cortex-M4F with high speed and low memory usage. They achieved this through having fast discrete Gaussian sampling and efficient polynomial multiplication. They found their implementation to be faster than ECC-based encryption schemes by one order of magnitude at minimum and about seven times faster than other R-LWE implementations. In [38], they propose an efficient R-LWE implementation on the ARM NEON and MSP430 architecture. They optimize the NTT-based polynomial multiplication and achieve an implementation that was faster than all other R-LWE implementations at the time. They also compared their implementation with classic cryptography schemes (RSA and ECC) and found that their optimization was also faster than these cryptoschemes. In [39] they propose an optimized implementation of Kyber and Dilithium for the Cortex-M4. The optimizations in the arithmetic resulted in increases of around 5% for Dilithium and just over 15% for Kyber.

One main area of concern for security when implementing cryptographic protocols is ensuring the security is not compromised by side channel attacks. This is where information such as duration, power or faults that occur can be used to undermine the security provided by the protocol. It is important to make sure that implementations are taking this vulnerability into account. In [40] they propose an optimized implementation of the supersingular isogeny key encapsulation mechanism (SIKE) for an ARM Cortex-M4. They measured the energy consumption, and also found whilst benchmarking their implementation that it was around 20% faster than previous implementations. It was found for the Cortex-M4 implementation of SIKE based on the reference implementation that a full recovery of keys was possible with a side-channel attack [41]. This brings to the fore the importance of implementation-specific security testing. In [42], they propose an architecture for fault detection that will provide protection against fault analysis vulnerabilities among R-LWE lattice-based key generation and encryption. This implementation was benchmarked on a field-programmable gate array (FPGA) and provided a high level of error coverage with a relatively low overhead.

In [43], they discuss the use of Curve448 to provide hybrid postquantum security. This means that the algorithm can provide both classical and postquantum cryptography. This provides a greater level of assurance, as it provides protection against quantum attacks, but if vulnerabilities are found in the postquantum protocol—which, given the recency of most postquantum protocols, is a notable concern for many—the classic cryptographic scheme is also there. They propose the first implementation of Curve448 on a 32-bit ARM Cortex-M4. They provide assurance against a timing attack with their regular and constant

implementation. In [44], they outline a design for Curve448 and Ed448 on Cortex-M4. These are curves that are expected to be able to provide post-quantum security. This design of the underlying operations has resulted in a significantly faster implementation for the key exchange and digital signature for Curve448 and Ed448, respectively.

In [45], they outline a new approach to implement a cryptographic accelerator on a FPGA for an Ed25519-based digital signature scheme. This is a hybrid digital signature scheme, and through its optimization, it resulted in an improved efficiency of around 84%.

A particular type of side-channel attack that is often a consideration for the IoT context is fault attacks. This is where errors occur during the running of the algorithm, causing vulnerabilities. In [46], they assess error resiliency and test on many platforms. In [47], they propose a scheme on an FPGA that provides high error coverage.

Another area for vulnerability is when the tools for side-channel attacks are used in conjunction. For instance, differential power analysis and differential fault analysis can be used together, and this results in the countermeasures in place not being effective. Some work has been conducted on this, such as in [48], where they suggest that an evolutionary cipher would remain secure against this type of attack.

These papers highlight the importance of optimizing implementations for a given platform. They also often do not consider the medium of communication, e.g., MQTT, and the effect this will have on the efficiency of the implementation. They also bring to the fore the importance of ensuring the implementations are assessed according to the security they provide against all types of attacks, including side-channel attacks, which can often be overlooked. This will need to be considered when further work is conducted to create a proper implementation of the KEM-authenticated MQTT protocol.

7. Conclusions and Future Work

In this paper, we have outlined the importance of using authentication with MQTT for IoT devices, especially in a postquantum world.

We have seen, throughout our work, that adding postquantum signature-based authentication to MQTT is likely to have acceptable overhead in the order of kilobytes, except memory usage in cases where the device is constrained. Postquantum KEM-based authentication for MQTT similarly has acceptable overhead, with the advantage of a much faster connection speed, making it particularly useful in contexts where swift and secure communication is essential.

Next steps would be to implement a full version of the KEM-based authentication in MQTT with one of the existing MQTT brokers, e.g., Mosquitto or wolfMQTT. The protocol could then be run on IoT devices to determine the overhead of authentication in practice. This could be carried out with the KEM algorithm CRYSTALS-Kyber and compared with postquantum digital signature algorithms such as CRYSTALS-Dilithium and Falcon. It is important that this implementation be assessed as to the security it provides against side-channel attacks such as timing, fault and power attacks.

Based on our findings, we believe that postquantum authentication is an important part of MQTT in IoT. We also believe that KEMs are a valuable alternative to digital signatures to provide authentication, because they can provide a more efficient interaction between clients and brokers.

Author Contributions: Conceptualization, C.G.; methodology, C.G. and J.S.; software, J.S.; validation, C.G. and J.S.; formal analysis, J.S.; investigation, J.S.; resources, C.G.; data curation, J.S.; writing—original draft preparation, J.S.; writing—review and editing, C.G. and J.S.; supervision, C.G.; project administration, C.G. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: Code used and data generated can be found in the GitHub repositories mentioned in the Implementation sections.

Acknowledgments: We would like to thank Raven Townsend for his support in writing the Python skeleton code for the KEM-based authentication.

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

Abbreviations

The following abbreviations are used in this manuscript:

ABE	Attribute-Based Encryption
AES	Advanced Encryption Standard
ARM	Advanced RISC Machines
CPU	Central Processing Unit
CRYSTALS	Cryptographic Suite for Algebraic Lattices
DSA	Digital Signature Algorithm
DTLS	Datagram Transport Layer Security
ECC	Elliptic Curve Cryptogrophy
FPGA	Field-Programmable Gate Array
HMAC	Hash Message Authenticated Code
IoT	Internet of Things
IPSec	Internet Protocol Security
KEM	Key Encapsulation Mechanism
KEMTLS	Key Encapsulation Mechanism authenticated Transport Layer Security.
MQTT	Message Queue Telemetry Transport
MQTT-SN	Message Queue Telemetry Transport for Sensor Networks
NIST	National Institute of Standards and Technology
PKI	Public Key Infrastructure
R-LWE	Ring-Learning With Errors
RSA	Rivest–Shamir–Adleman
SHA	Secure Hash Algorithm
SIKE	Supersingular Isogeny Key Encapsulation mechanism
SSL	Secure Sockets Layer
TCP	Transmission Control Protocol
TLS	Transport Layer Security
UDP	User Datagram Protocol
VPN	Virtual Private Network

Notation Used in Figures

MQTT Handshake with Digital Signature

ct_s	Encapsulated Premaster Secret
pk_s	Public Key for Client
pms	Premaster Secret
r_b	Random Nonce Sent by Broker
r_c	Random Nonce Sent by Client
sk_s	Secret Key

MQTT Handshake with KEM

ct_s	Encapsulated Premaster Secret
pk_e	Ephemeral Public Key
r_b	Random Nonce Sent by Broker
r_c	Random Nonce Sent by Client
sk_e	Ephemeral Secret Key
sk_s	Secret Key
ss_e	Ephemeral Shared Secret
ss_s	Static Shared Secret

References

1. Dierks, T.; Rescorla, E. The Transport Layer Security (TLS) Protocol Version 1.2. *RFC* **2008**, *5246*, 1–10.
2. Frankel, S.; Kent, K.; Lewkowski, R.; Orebaugh, A.D.; Ritchey, R.W.; Sharma, S.R. *Guide to IPsec VPNs*; US Department of Commerce, Technology Administration, National Institute of Standards and Technology: Gaithersburg, MD, USA, 2005.
3. Perlman, R. An overview of PKI trust models. *IEEE Netw.* **1999**, *13*, 38–43. [CrossRef]
4. Barker, E. Digital Signature Standard (DSS). In *Federal Information Processing Standards Publication (FIPS)*; FIPS 186-4; US Department of Commerce, Technology Administration, National Institute of Standards and Technology: Gaithersburg, MD, USA, 2013. [CrossRef]

5. Dworkin, M.; Barker, E.; Nechvatal, J.; Foti, J.; Bassham, L.; Roback, E.; Dray, J. Advanced Encryption Standard (AES). In *Federal Information Processing Standards Publication*; SP 800-56A Rev. 3; US Department of Commerce, Technology Administration, National Institute of Standards and Technology: Gaithersburg, MD, USA, 2001. [CrossRef]
6. Shor, P.W. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Rev.* **1999**, *41*, 303–332. [CrossRef]
7. Jozsa, R. Searching in Grover's algorithm. *arXiv* **1999**, arXiv:quant-ph/9901021.
8. Grassl, M.; Langenberg, B.; Roetteler, M.; Steinwandt, R. Applying Grover's algorithm to AES: Quantum resource estimates. In Proceedings of the Post-Quantum Cryptography: 7th International Workshop, PQCrypto 2016, Fukuoka, Japan, 24–26 February 2016; Proceedings 7; Springer: Cham, Switzerland, 2016; pp. 29–43.
9. Wallden, P.; Kashefi, E. Cyber security in the quantum era. *Commun. ACM* **2019**, *62*, 120. [CrossRef]
10. Hazan, E; Ménard, A; Ostojic, I; Pate, M. *The Next Tech Revolution: Quantum Computing*; Technical Report; McKinsey & Company: Paris, France, 2020.
11. Denchev, V.S.; Boixo, S.; Isakov, S.V.; Ding, N.; Babbush, R.; Smelyanskiy, V.; Martinis, J.; Neven, H. What is the Computational Value of Finite-Range Tunneling? *Phys. Rev. X* **2016**, *6*, 031015. [CrossRef]
12. Suresh, P.; Daniel, J.V.; Parthasarathy, V.; Aswathy, R. A state of the art review on the Internet of Things (IoT) history, technology and fields of deployment. In Proceedings of the 2014 International Conference on Science Engineering and Management Research (ICSEMR), Chennai, India, 27–29 November 2014; pp. 1–8.
13. Cheng, C.; Lu, R.; Petzoldt, A.; Takagi, T. Securing the Internet of Things in a quantum world. *IEEE Commun. Mag.* **2017**, *55*, 116–120. [CrossRef]
14. Benantar, M. The Internet public key infrastructure. *IBM Syst. J.* **2001**, *40*, 648–665. [CrossRef]
15. Weise, J. Public key infrastructure overview. In *Sun BluePrints OnLine—August 2001*; Sun Microsystems, Inc.: Palo Alto, CA, USA, 2001; pp. 1–27.
16. Bos, J.; Ducas, L.; Kiltz, E.; Lepoint, T.; Lyubashevsky, V.; Schanck, J.M.; Schwabe, P.; Seiler, G.; Stehle, D. CRYSTALS-Kyber: A CCA-Secure Module-Lattice-Based KEM. In Proceedings of the 2018 IEEE European Symposium on Security and Privacy, London, UK, 24–26 April 2018; pp. 353–367.
17. Nist, C. The digital signature standard. *Commun. ACM* **1992**, *35*, 36–40. [CrossRef]
18. Merkle, R.C. A digital signature based on a conventional encryption function. In Proceedings of the Conference on the Theory and Application of Cryptographic Techniques, Santa Barbara, CA, USA, 16–20 August 1987; Springer: Berlin/Heidelberg, Germany, 1987; pp. 369–378.
19. Buchmann, J.; Dahmen, E.; Szydlo, M. Hash-based digital signature schemes. In *Post-Quantum Cryptography*; Springer: Berlin/Heidelberg, Germany, 2009; pp. 35–93.
20. Dobraunig, C.; Eichlseder, M.; Mendel, F.; Schläffer, M. Ascon: Lightweight Authenticated Encryption & Hashing. Available online: https://ascon.iaik.tugraz.at/ (accessed on 14 July 2023).
21. Boutin, C. NIST Selects "Lightweight Cryptography" Algorithms to Protect Small Devices. 2023. Available online: https://www.nist.gov/news-events/news/2023/02/nist-selects-lightweight-cryptography-algorithms-protect-small-devices (accessed on 14 July 2023).
22. Kittur, A.S.; Jain, A.; Pais, A.R. Fast verification of digital signatures in IoT. In Proceedings of the International Symposium on Security in Computing and Communication, Manipal, India, 13–16 September 2017; Springer: Singapore, 2017; pp. 16–27.
23. Schwabe, P.; Stebila, D.; Wiggers, T. Post-quantum TLS without handshake signatures. In Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security, Virtual, 9–13 November 2020; pp. 1461–1480.
24. Computer Security Division Information Technology Laboratory. *Post-Quantum Cryptography Standardization—CSRC*; NIST: Gaithersburg, MD, USA. 2017.
25. Karabulut, E.; Aysu, A. FALCON Down: Breaking FALCON Post-Quantum Signature Scheme through Side-Channel Attacks. In Proceedings of the 58th ACM/IEEE Design Automation Conference, San Francisco, CA, USA, 5–9 December 2021; pp. 691–696.
26. Perrone, G.; Vecchio, M.; Pecori, R.; Giaffreda, R. The Day After Mirai: A Survey on MQTT Security Solutions after the Largest Cyber-attack Carried Out through an Army of IoT Devices. In Proceedings of the IoTBDS, Porto, Portugal, 24–26 April 2017; pp. 246–253.
27. Singh, M.; Rajan, M.; Shivraj, V.; Balamuralidhar, P. Secure MQTT for Internet of Things (IoT). In Proceedings of the 2015 Fifth International Conference on Communication Systems and Network Technologies, Gwalior, India, 4–6 April 2015; pp. 746–751. [CrossRef]
28. Wang, X.; Zhang, J.; Schooler, E.M.; Ion, M. Performance evaluation of attribute-based encryption: Toward data privacy in the IoT. In Proceedings of the 2014 IEEE International Conference on Communications (ICC), Sydney, NSW, Australia, 10–14 June 2014; IEEE: Piscataway, NJ, USA, 2014; pp. 725–730.
29. Goyal, V.; Pandey, O.; Sahai, A.; Waters, B. Attribute-based encryption for fine-grained access control of encrypted data. In Proceedings of the 13th ACM Conference on Computer and Communications Security, Alexandria, VA, USA, 30 October–3 November 2006; pp. 89–98.
30. Park, C.S.; Nam, H.M. Security architecture and protocols for secure MQTT-SN. *IEEE Access* **2020**, *8*, 226422–226436. [CrossRef]
31. Amaran, M.; Rohmad, M.; Adnan, L.; Mohamed, N.; Hashim, H. Lightweight security for mqtt-sn. *Int. J. Eng. Technol.* **2018**, *7*, 223–226. [CrossRef]

32. Haakegaard, R.; Lang, J. The Elliptic Curve Diffie-Hellman (ecdh). 2015. Available online https://koclab.cs.ucsb.edu/teaching/ecc/project/2015Projects/Haakegaard+Lang.pdf (accessed on 14 July 2023).
33. Wu, W.; Zhang, L. LBlock: A lightweight block cipher. In Proceedings of the International Conference on Applied Cryptography and Network Security, Nerja, Spain, 7–10 June 2011; Springer: Berlin/Heidelberg, Germany, 2011; pp. 327–344.
34. Chung, C.C.; Pai, C.C.; Ching, F.S.; Wang, C.; Chen, L.J. 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, Portland, OR, USA, 27 June–1 July 2022; pp. 525–526.
35. Septien-Hernandez, J.A.; Arellano-Vazquez, M.; Contreras-Cruz, M.A.; Ramirez-Paredes, J.P. A Comparative study of post-quantum cryptosystems for Internet-of-Things applications. *Sensors* **2022**, *22*, 489. [CrossRef] [PubMed]
36. Shahbazi, K.; Ko, S.B. Area and power efficient post-quantum cryptosystem for IoT resource-constrained devices. *Microprocess. Microsyst.* **2021**, *84*, 104280. [CrossRef]
37. De Clercq, R.; Roy, S.S.; Vercauteren, F.; Verbauwhede, I. Efficient software implementation of ring-LWE encryption. In Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition (DATE), Grenoble, France, 9–13 March 2015; IEEE: Piscataway, NJ, USA, 2015; pp. 339–344.
38. Liu, Z.; Azarderakhsh, R.; Kim, H.; Seo, H. Efficient software implementation of ring-LWE encryption on IoT processors. *IEEE Trans. Comput.* **2017**, *69*, 1424–1433. [CrossRef]
39. Abdulrahman, A.; Hwang, V.; Kannwischer, M.J.; Sprenkels, A. Faster kyber and dilithium on the cortex-m4. In Proceedings of the Applied Cryptography and Network Security: 20th International Conference, ACNS 2022, Rome, Italy, 20–23 June 2022, Springer: Cham, Switzerland, 2022; pp. 853–871.
40. Anastasova, M.; Azarderakhsh, R.; Kermani, M.M. Fast strategies for the implementation of SIKE round 3 on ARM Cortex-M4. *IEEE Trans. Circuits Syst. I Regul. Pap.* **2021**, *68*, 4129–4141. [CrossRef]
41. Genêt, A.; de Guertechin, N.L.; Kaluđerović, N. Full key recovery side-channel attack against ephemeral SIKE on the Cortex-M4. In Proceedings of the Constructive Side-Channel Analysis and Secure Design: 12th International Workshop, COSADE 2021, Lugano, Switzerland, 25–27 October 2021; Proceedings 12; Springer: Cham, Switzerland, 2021; pp. 228–254.
42. Sarker, A.; Kermani, M.M.; Azarderakhsh, R. Fault detection architectures for inverted binary ring-LWE construction benchmarked on FPGA. *IEEE Trans. Circuits Syst. II Express Briefs* **2020**, *68*, 1403–1407. [CrossRef]
43. Seo, H.; Azarderakhsh, R. Curve448 on 32-bit arm cortex-m4. In Proceedings of the Information Security and Cryptology–ICISC 2020: 23rd International Conference, Seoul, Republic of Korea, 2–4 December 2020; Proceedings 23; Springer: Cham, Switzerland, 2021; pp. 125–139.
44. Anastasova, M.; Azarderakhsh, R.; Kermani, M.M.; Beshaj, L. Time-Efficient Finite Field Microarchitecture Design for Curve448 and Ed448 on Cortex-M4. In Proceedings of the Information Security and Cryptology–ICISC 2022: 25th International Conference, ICISC 2022, Seoul, Republic of Korea, 30 November–2 December 2022; Revised Selected Papers; Springer: Cham, Switzerland, 2023; pp. 292–314.
45. Bisheh-Niasar, M.; Azarderakhsh, R.; Mozaffari-Kermani, M. Cryptographic accelerators for digital signature based on Ed25519. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* **2021**, *29*, 1297–1305. [CrossRef]
46. Mozaffari-Kermani, M.; Azarderakhsh, R.; Aghaie, A. Reliable and error detection architectures of Pomaranch for false-alarm-sensitive cryptographic applications. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* **2015**, *23*, 2804–2812. [CrossRef]
47. Ahir, P.; Mozaffari-Kermani, M.; Azarderakhsh, R. Lightweight architectures for reliable and fault detection Simon and Speck cryptographic algorithms on FPGA. *ACM Trans. Embed. Comput. Syst. (TECS)* **2017**, *16*, 1–17. [CrossRef]
48. Tang, M.; Qiu, Z.; Yang, M.; Cheng, P.; Gao, S.; Liu, S.; Meng, Q. Evolutionary ciphers against differential power analysis and differential fault analysis. *Sci. China Inf. Sci.* **2012**, *55*, 2555–2569. [CrossRef]