Incorporating post-quantum cryptography in a microservice environment

D.L. Weller and R. van der Gaag

Security and Network Engineering - University of Amsterdam

Supervisors (Deloitte): Itan Barmes, Bartosz Czaszynski and Ruud Schellekens

February 9, 2020

Abstract - Nowadays secure connections are established with modern cryptography. The modern cryptography algorithms are considered secure due to their infeasible decryption time if one is not in possession of the private key. With a quantum computer which is able to run, for example, Shor's algorithm, it is likely that the classical cryptography is cracked within feasible time. In this research, different Post-Quantum Cryptography (PQC) algorithms are tested on performance of the certificate generation, certificate signing and performing a TLS handshake with mutual authentication. PQC is a family of cryptography algorithms which cannot be cracked in a feasible time for a Quantum computer but still can be used with classical computers. One of the areas cryptography is used are microservice architectures. Microservice architectures often include a large number of secured connections and the transition to PQC may be a challenge. This research focuses on the practical feasibility of using PQC in a microservice architecture. We conclude that the performance of PQC is similar to the performance of classical cryptography. The main differences is seen between lattice-based (which performs best), code-based (which performs slightly worse) and isogeny-based (which is the least performing) algorithms. For switching from classical cryptography to PQC, a hybrid solution is in place. The performance of the hybrid solution is as good as the slowest algorithm used within this hybrid solution. Since a hybrid solution does take the sum of both cryptography algorithms, this is a feasible solution for transitioning to PQC.

1. Introduction

Cryptography plays a large part in everyday life. It is used by numerous services ranging from mobile communications, website security and protecting sensitive data from prying eyes. Nowadays classical cryptography (e.g. RSA) is used to secure connections. This is considered safe due to the long decryption times if one is not in possession of the key, making this approach infeasible for attackers.

According to [1] gate-based quantum computers pose a significant threat against asymmetric encryption. A gate-based quantum computer could be used to run, for example, Shor's algorithm. **W. Buchanan et. al** also states that it is likely that this quantum superiority will become a practical

threat within 10 years. Asymmetric encryption is used for a lot of communication which build upon their security (e.g. TLS, SSH, WPA(2), DNSSec, IKEv2, S/MIME).

In order to stay ahead of this threat preparations are needed for a migration from classical cryptography (e.g. RSA) to Post-Quantum Cryptography (PQC). The goal of post-quantum cryptography (also called quantum-resistant cryptography) is to develop cryptosystems that are secure against both quantum and classical computers. For a smooth transition to PQC algorithms, interoperability with existing communication protocols and networks is often necessary. Especially in large enterprise companies this will prove to be a challenge.

Scalable information exchange systems as used by large corporations do not only require proper security, but often have performance requirements as well. One of software components used in such systems is *Apache Kafka*, a platform designed for real-time distributed messaging on an enterprise scale [2]. This platform consists of multiple message brokers, each having multiple connections to micro-services. At the time of writing, most connections between the micro-services and the message brokers are secured based on classical cryptographic algorithms.

2. Research Questions

When introducing new cryptographic algorithms in microservice environments, scalability and ease of implementation are important requirements. In addition, feasibility of the use of post-quantum cryptography (PQC) algorithms in enterprise environments is still unknown. Consequently, this project will focus on the following research question:

What are the implications of transitioning to Post-Quantum Cryptography in microservice architectures where public key infrastructure is used for key establishment and certificate signing?

In order to assist the main research question, the following sub-questions are defined:

- What performance differences are observed with different Post-Quantum Cryptography in practice?
- How feasible is practical transitioning from classical cryptography to Post-Quantum Cryptography?

In the next section we will expound further upon PQC and related projects, as well as the distributed messaging systems and in particular Apache Kafka. Then, our methodology is described in section 4. In addition, an attempt at integration of PQC in Apache Kafka will be elaborated upon. Next we will provide the results of our experiments and elaborate on our findings in section 5, and the discussion regarding the results in section 6. Lastly, our conclusions are described in section 7 and the paper is concluded with the proposed future work in section 8.

3. Related Work

In this section the related projects to PQC and enterprise messaging systems are described. The primary development of PQC algorithms is driven by the standardisation project at the National Institute of Standards and Technology (NIST). A brief overview of the different categories of PQC is given. Some of these algorithms have been implemented by the Open Quantum Safe (OQS) project. Finally, the distributed message broker Apache Kafka which the PoC will be based upon is described.

3.1. Post-Quantum Cryptography

3.1.1. National Institute of Standards and Technology

The NIST has initiated a project focused on standardising post-quantum cryptography (PQC) algorithms [3]. This project provides a platform for quantum-resistant public-key cryptography algorithm

proposals. This is done in three rounds of peer reviewing the proposed algorithms, narrowing down potential candidates each round. As of this writing, the second round of review is ongoing and on January 30th 2019, NIST has announced 26 candidates for standardization [4]. Of these 26 participants, there are 17 public-key encryption and key-establishment algorithms and 9 digital signature algorithms. Each of these proposals include parameter sets that are related to one of the by NIST specified security levels. An overview and a short description of these security levels is presented in Figure 1.

Table 1: Security Assurance Levels defined by NIST [3]

Level	**Security Description**
I	At least as hard to break as AES128 (exhaustive key search)
II	At least as hard to break as SHA256 (collision search)
III	At least as hard to break as AES192 (exhaustive key search)
IV	At least as hard to break as SHA384 (collision search)
V	At least as hard to break as AES256 (exhaustive key search)

3.1.2. Open Quantum Safe

The Open Quantum Safe (OQS) project revolves around the implementation of quantum-safe algorithms [5]. The main contribution of this project is the development of open source prototype libraries implementing a subset of the proposed algorithms of the NIST post-quantum project into forks of well known cryptography software stacks, such as OpenSSL and OpenSSH [6][7]. This is done by bundling implementations into the open source library named *liboqs*. This library is focused on providing a single interface for other libraries to build on. A subset of these implementations is provided by the PQClean project [8]. From there, the OQS OpenSSL fork may be used to create TLS sessions based on PQC provided by the liboqs project.

3.1.3. PQC algorithm categories

There are 5 different categories of PQC algorithms. Most of these algorithms rely on the difficulty of specific mathematical problems. In addition to that there are PQC algorithms, specifically digital signature algorithms, that are hash function based. Each category contains different properties due to the difference in their underlying structures. In the next sections we summarise related work on these categories.

Lattice-based The underlying mathematical structure of the algorithms in this category consist of discrete mathematical structures called *lattices* [3][9]. One of the main hardness problems used in lattice cryptography is the Shortest Vector Problem, which is to find the shortest vector to the nearest lattice point. The strength of the security of these algorithms is dependent on the lattice bases that are used. In other words, there are specific families of lattices that provide variable levels of security. Some of these lattice families are provably secure under a worst-case hardness assumption.

Lattices have also been used to develop signature schemes [9]. One of the most promising algorithms is BLISS (Bimodal Lattice Signature Scheme), especially due to the relatively efficient implementation compared to classical algorithms such as RSA and ECDSA. This also holds true for key establishment mechanisms described previously and is the main reason for the increased scientific interest in this category.

Code-based Stemming from one of the oldest post-quantum proposals [10], the algorithms in this category are based on well known hardness problems of decoding error corrected messages without having access to the decoder information [11]. Although the key encapsulation mechanism and encryption processes have been found to be faster than classical cryptography alternatives, the generation time and the space required for the keys strongly contrasts these benefits [12]. However, there are some NIST submissions that make use of this basis and achieve good results [4].

Multivariate polynomial cryptography Algorithms in this category are based on the difficulty of solving Multivariate-quadratic problems [13]. These algorithms can be used for encryption and signing. However, signature creation and verification is the main area where these algorithms are used. Recent developments allow for very small private keys and signature sizes [12], though public key sizes remain relatively large. As of this moment there are four submissions to the NIST competition that are multivariate-based.

Super-singular elliptic curve isogenies This category consist of key establishment algorithms based on super-singular isogeny graphs of elliptic curves. The security relies on the difficulty of finding the isogeny mapping between the elliptic curves. In 2011 the first proposal was published utilising this basis [14], and as such this field of research relatively new. The main advantage are the small key sizes and perfect forward secrecy. Perfect forward secrecy is, in short, the ability to keep past communications safe from a compromised private key. In contrast to these advantages, the computations needed to establish a shared key are very intensive and thus computation times are very high. Also, being a relatively new and complex field of research, a lot of questions remain unanswered about the security of these algorithms.

Hash-based Lastly, this category revolves around signature algorithms based on hash functions. As RSA based signature schemes are useless in a post-quantum world, the interest in alternatives shifted towards a well known Merkle tree-based signature schemes. Merkle trees, also known as Hash trees, are tree like structures that are used to securely verify data. One of the problems with Merkle tree-based schemes is the need for keeping state. In addition, security of the scheme is compromised if a single error is made in keeping this state. In the most common situation this would mean that a key is used multiple times to sign data. The most recent proposal in the NIST competition building on this related work, named SPHINCS+, proposes a stateless scheme [15]. This would mean that the signee is relieved of the burden of keeping administration of what keys have been used and which are unused. Signing times and signature sizes are relatively large, though the public keys are considered very small with this implementation [12].

3.2. Transport Layer Security 1.3

The Transport Layer Security (TLS) protocol was defined to create an up-to-date standard for secure communications, being the successor of the Secure Socket Layer protocol from the early nineties. The newest version is TLS 1.3 formally introduced as a standard in 2018, ten years after the introduction of TLS 1.2 [16][17]. TLS 1.3 contains many performance improvements over the previous version. For example, different handshake modes are introduced such as *sessions resumption* and *zero round-trip time*, among others. For brevity, we will focus in this paper on the parts influenced by PQC.

The default mode for TLS 1.3 is the *full handshake* mode which consist of four communication steps. These steps are also presented in Figure 1. First, the client initiates the connection with a message containing *client hello* and a *key share* extension fields. In this message, the client indicates the PQC signature scheme and Key Encapsulation Mechanism (KEM) that it supports in the key share extension. Then, the server responds with a message containing *server hello*, *key share*, *encrypted extensions* and *certificate request* fields. These fields contain the KEM information and the certificate from the server. Optionally the server can require authentication of the client. In addition, the server also sends the *finished* message, indicating the confirmation of the handshake. The transcript of all communication is signed by the server as well and sent to the client for verification. In the third step the client sends back its certificate and the signature to authenticate itself. As stated in the previous step, this is optional

and only used when mutual authentication is required by the server. The last step marks the end of the TLS handshake and communication of application data is enabled using the shared key.

In these four steps two different mechanism are used that are influenced by the use of PQC: key exchange and certificate verification. In addition to the shorter handshake time, TLS 1.3 introduces a multitude of new features and changes. On of these changes is a new communication mode named zero round trip time (0-RTT) and is based on a previously established sessions. The pre-shared key (PSK) that was established can be used to immediately resume a session by sending it in advance to the server. The server then decides if the PSK is valid and resumes the session or a new full handshake may be requested.

Other changes include the addition of elliptic curve based KEMs to the standard specification and the redesign of key derivation function. In addition, symmetric encryption algorithms that are considered legacy have been removed from the list of allowed cipher suites. Though, in this research we will mainly focus on the full TLS 1.3 handshake.

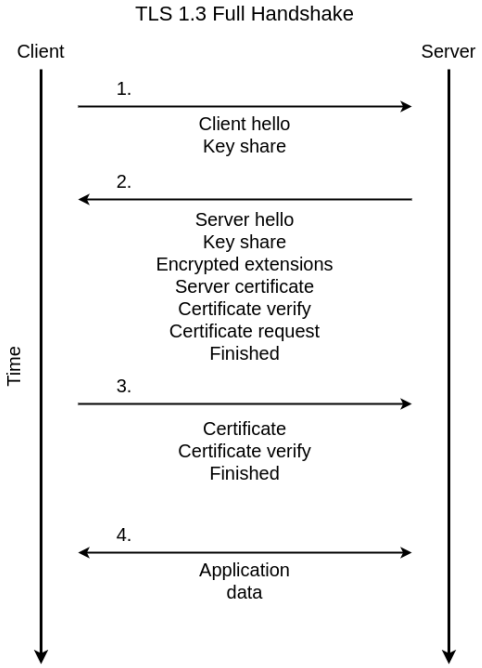


Figure 1: The four steps of the TLS 1.3 full handshake.

3.3. Message busses

In microservice architectures, the message bus is a common way to create a scalable communication interface separate from microservice development. The message bus often consist of multiple message brokers. A message broker is an intermediary module which can be used to translate messages of the sender protocol to the protocol of the receiver. Message brokers also are a solution for an architecture where one-to-many communication is happening.

For message broker-systems, there are a set of guarantees which have to be taken into account for selecting a suitable broker [18]:

Correctness
The property of correctness can be defined with three primitives: no-loss, no-duplication, no-disorder. Thereby these primitives can be discussed with Delivery Guarantees (e.g. at most once, at least once and once and only once) and Ordering Guarantees (e.g. no ordering, partitioned ordering and global ordering).

Availability
The availability of the system is the degree to which the system is able to maximize its up-time. In context of PQC TLS sessions, the main point of attention is the ability to maintain a stable system under different handshake loads. This includes key encapsulation mechanism as well as the signature verification times.

Transactions
To increase performance transactions may be bundled into into single units that are sent to or from the message bus. This has no impact on the creation of TLS sessions as mainly the data transport part of the session can be impacted by larger packets. However, large packets over a unstable network may impact the availability of the message brokers, for when a large message is lost it needs to be reset again. If the connection becomes congested due to a (bad) configuration, handshakes may have to be reset which can be a costly operation.

Scalability
Defined as the ability of a system to evolve, scalability of the system in term of TLS built on PKI mainly involves the certificate generation and validation. In order to spin up new microservices, certificates have to be generated in a certain amount of time. However, a continuous process can be setup up to accommodate for spikes. In addition, handshakes should be scalable as well. This mainly comes down to the time each handshake will take, and computationally intensive this process is compared to the amount of new handshakes coming in.

Efficiency
Efficiency is commonly measured in two ways, *latency* (or response time), and *throughput* (or bandwidth)

- Latency is the elapsed time a program takes to perform its task. In terms of networking, this is often also called *round trip time*. Symmetric encryption has a noticeable effect on the latency of a program, as it introduces extra overhead of encrypting and decrypting packets. TLS session setup is less influenced by the extra overhead as the sessions are typically used over longer periods of time.

- Throughput is often determined by the number of packets (or alternatively, bytes) per time unit that can be transported between clients. This could be enhanced by adding additional resources in parallel.

3.3.1. Kafka

In 2011, an in-house project at LinkedIn was published under the name of *Kafka* [2] by Kreps et al. Kafka was developed to handle the increasing volume of data that is internally processed by enterprise companies. Existing technologies did not have sufficient performance for communicating log messages of internal (micro)services in a distributed environment. A few predecessors are described such as Facebook's Scribe and Yahoo's data highway project [19][2]. The main difference of Kafka's implementation and its predecessors is the fact that the message brokers are *passive*. This means that all message communication from and to the bus is initiated by the producers and consumers of those messages, allowing the message flow to be regulated by the (micro)services itself. In addition, the message brokers do not need to keep states of each sessions and are thus able to focus on keeping message states, increasing scalability. Soon after the initial publication in 2011, Kafka became part of the Apache Software Foundation and is now also known as *Apache Kafka*. In this paper we use the terms Kafka and Apache Kafka interchangeably.

Apache Kafka supports encryption and mutual authentication for communication between the bus and the micro-services with use of certificates, based on classical cryptography [20]. Figure 2 illustrates a basic overview of the communication between Kafka and the micro-services. As Kafka is built with Java the TLS sessions used in these communication lines are based on the The Java Cryptography Architecture.

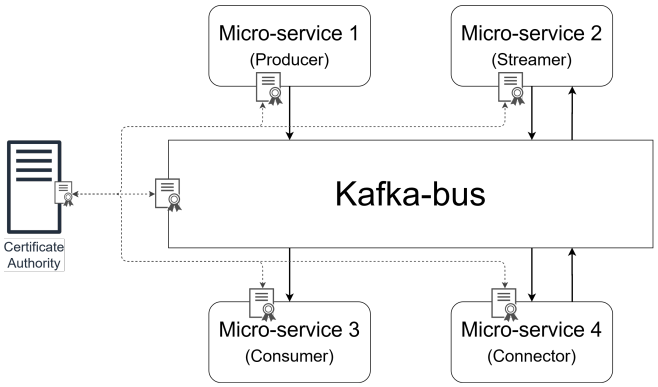


Figure 2: Basic Kafka environment with certificate-based encryption and authentication.

3.3.2. Java Cryptography Architecture

The Java Cryptography Architecture (JCA) is a framework for Java developers to develop secure applications. Multiple Application Programming Interfaces (API) are defined under the JCA to provide different services. The JCA also specifies a unified way of introducing new *cryptographic service providers*. These providers can be used to access different implementations by all interfaces throughout the JCA specification. Figure 3 provides an overview of the provider architecture.

For this research two APIs are relevant: the Java Cryptography Extension (JCE) and the Java Secure Socket Extensions (JSSE). The JCE is used to access cryptography implementations which can be used in applications directly or utilised by protocols such as TLS. This is where the second API comes in, the JSSE. JSSE is used to create SSL and TLS sessions in Java applications. This interface is used by Kafka to create secure connections between the microservices and the message brokers. Key pairs and certificates may be built using the JCE interface and are stored in Java Key Stores (JKS). Both the JCE and JSSE come with a set of standard implementations built in Java.

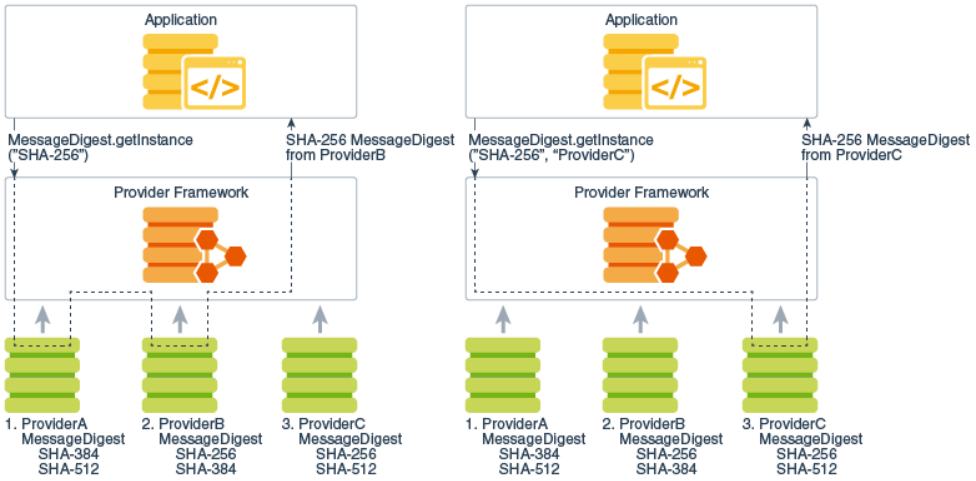


Figure 3: Example of the operation of the Java Cryptography Architecture, originating from the official JCA documentation [21].

3.3.3. Kafka & PQC providers

Currently there are three different implementation libraries that include a subset of PQC algorithms from the NIST competition:

- PQCRYPTO - libpqcrypto
- Open Quantum Safe - liboqs
- Bouncy Castle

The PQCRYPTO project is a collaboration of numerous cryptography scientists around the world, bundling their efforts in designing PQC [8]. The libpqcrypto library provides the implementations of these algorithms under a unified interface. Although native C and Python interfaces for the cryptography algorithms are defined, no TLS interface is available in this project. Liboqs is a project by OQS and focuses on bundling C implementations of the round 2 NIST candidates [22]. The OpenSSL fork by OQS makes use of liboqs library and provides a TLS interface. Bouncy Castle (BC) is a project focused on cryptography implementations for Java and C# [23]. BC has both JCE and JSSE interfaces available.

Of these three, libpqcrypto is the most complete, including 19 of the 22 NIST submissions, although no TLS projects have been built on this library yet. BC includes only qTesla, SPHINCS and Newhope implementations, and has no TLS API available. OQS on the other hand has started with the development of a TLS API as previously mentioned. The OQS OpenSSL fork contains 14 NIST submissions at the moment of writing, and is the only library providing an experimental TLS API [22]. Therefore this research will focus on the implementation of the OQS OpenSSL fork in the PoC.

4. Methodology

In this research multiple post-quantum cryptography algorithms are investigated and their implementations in practice are evaluated. A proof of concept will be built to demonstrate the findings. In the following subsections we describe the approach to provide answers to our research questions, the proof of concept and the scope of this research.

4.1. Proof of Concept & Performance Measurements

The first part of this research is focused on setting up a test environment based on PQC and evaluating the practical feasibility thereof. In order to do this, a Proof of Concept (PoC) is built for implementing PQC algorithms in a microservice architecture. Specifically, the message bus system Apache Kafka is used to represent the communication channels within such an architecture. The OpenSSL fork provided by OQS was chosen for generating and signing the certificates. This choice was made since the OpenSSL fork is the only TLS library available at the time of writing, and creating a TLS interface from scratch is out of scope. For Apache Kafka to be able to use OpenSSL, a third party wrapper is needed. This is due to the fact that Apache Kafka makes use of the JSSE architecture, which is Java based, whereas OpenSSL is C based.

For each combination of available signature and key establishment algorithms available in the OQS OpenSSL library, elapsed time and peak heap memory required to complete TLS 1.3 handshakes are measured. There are three different setups of algorithms: PQC, non-PQC and hybrid. Especially the algorithm combinations within the same setup are focused on. The measurement take place on two different servers, one acting as a TLS 1.3 server and the other as a TLS 1.3 client. In order to to identify the added latency of the network itself, we also measured the network latency. In addition, server hardware specifications can be found in appendix A.3.

Each signature and key establishment algorithm combination is then performed 500 times to produce reliable means and standard deviations. These measurements are performed by the command line tool named Hyperfine [24]. In addition, each algorithm combination is profiled by Heaptrack to provide insight in the memory consumption difference of each algorithm combination [25]. From these memory profiles the peak heap memory is extracted. Last, the results are presented and evaluated per NIST defined security level.

4.2. Practical feasibility evaluation

The second part of this research is aimed at practical feasibility of the transitioning to PQC with modern microservices. In order to be able evaluate the practical feasibility we make a set of considerations on the microservice environment.

The first consideration is the degree the system is able to respond to fluctuations in demand. Usually microservice architectures are built with components that focus on a single task. This increases the scalability of the system: if more service apps are required by an increase in demand, more (often virtual) devices will be enabled to handle the extra system load. This means that multiple microservices need to be enabled within a certain time-window depending on the increase in demand, without causing instabilities in the system. When creating the new devices, new key pairs and certificates need to be generated as well in order to setup secured TLS sessions. The time required to generate key pairs, certificates and signatures may have a large impact on this process and is therefore one of the points of discussion.

The second consideration revolves around TLS session setup and maintenance. Once the certificate is generated and signed by the Certificate Authority, communication can be set up with the message bus. A session between the message bus and the client needs to be established with a TLS 1.3 handshake. Since further communication takes place with symmetric encryption, we scope down to the session establishment. Since the session establishment also impacts the time it takes before message bus and client can communicate, this will impact the practical feasibility of different algorithms. TLS 1.3 has many features that speed up the (re)connection of clients to servers. In order to guarantee that each test is performed with a full handshake, specific OpenSSL commands for setting up the server and client need specific instructions for the (debugging of) different operations are used [26]. This way we can guarantee that no zero round trip times (0-RTT) or session reuse is communicated, and thus each run represents a full handshake. In addition, packet captures will be generated for each run to validate the communication.

5. Results

5.1. Kafka PQC

As previously described, Kafka is built with Java and makes use of the JCA when setting up TLS connections. These TLS connections are by default reliant on the (classical) cryptography providers available in the JRE. In order to add OpenSSL to the list of these providers, a conversion of API calls is needed from the Java environment to the native C environment. There have been previous attempts to bridge the two, one of which is a sub-project of the Wildfly organisation [27]. This project, named Wildfly-OpenSSL, mainly focused on converting the TLS suite names between the environments, running on its own native C extension library in order to pass the library calls to the OpenSSL API. In this research we attempted to adapt the Wildfly-OpenSSL codebase to create a security provider integrating PQC implementations into JSSE, but unfortunately we did not manage to reach successful integration. The Wildfly-OpenSSL project is based on the older Java 8 which does not support TLS 1.3. This can however be updated to work on newer Java versions. The C library included in the Wildfly-OpenSSL project also has to be updated to accommodate for the changes in the newer OpenSSL library. The problem is that the API of the OQS OpenSSL fork has not been fully developed yet. As a result, no PQC TLS sessions could be established using the Wildfly-OpenSSL library. However, although development of the Wildfly-OpenSSL has ceased, this approach could be used to create a bridge between Apache Kafka and OpenSSL in the future.

The second option was to try the security provider available by The Legion of the Bouncy Castle project [23]. After a successful installation of the JCE security provider the PQC algorithms were detected in the JRE. Applications making use of the JCA are able to recognize the algorithms as an option, though the Java keytool used to create the keys and certificates in the necessary format did not function properly. Bouncy Castle Key Stores (BKS) can be generated with this tool, however no key or certificates could be added due to alleged corruption of the key store. Even when key stores with the required keys and certificates could successfully be generated, there is still no TLS provider that incorporates the required algorithms to actually make use of the key store.

As stated in the methodology the development of such libraries is considered out of scope. Consequently we opted to utilise the OQS OpenSSL server and client applications for our experiments, described in the next section.

5.2. OpenSSL PQC

The OQS OpenSSL project provides two application options to setup a testing environment for TLS connections based on the included PQC algorithms. Setting up the experiments consist of three parts: key pair and certificate generation, certificate signing, and the TLS handshake. First, the Certificate Authority (CA) is created by generating its key pair and certificate. Then, the server key pair and certificate are generated and signed by the CA. These two processes will be measured. Then, the client key pair and certificate are generated and signed by the CA as well to allow mutual authentication in the TLS handshake. The CA is added to the trust store of the server and the client. The server is started with all possible algorithms enabled of the category being measured (PQC, non-PQC and hybrid). From there the client initiates the handshake with a specific key encapsulation algorithm. All available key encapsulation algorithms in the category being measured are matched with each signature algorithm. When no combinations are available, less preffered key encapsulation algorithms are used. After the handshake is completed, the session is brought down immediately. The duration from the initiation of the handshake by the client up to the extermination of the session is measured.

The following section will present the results per experiment. A distinction is made between PQC, non-PQC and hybrid algorithm combinations. As there is a substantial number of algorithm combinations only the most interesting results will be displayed. For a full list of the results see appendix A.2.

5.3. Performance results of post-quantum cryptography algorithms

In this section the results are presented and discussed. The results are shown per classical cryptography (which functions as baseline) followed by the PQC and hybrid algorithm categories.

5.3.1. Generating key pairs & certificate requests

Since certificates needs to be generated, we measured the time it takes to generate a key pair with certificate requests. The time it takes to sign the certificates is discussed in the next section. We divided the results up in three different categories. First being the classical cryptography algorithms. These results function as the baseline. Second are the Post Quantum Cryptography algorithms. And the last category consists of Hybrid Post Quantum Cryptography.

Classical cryptography As can be seen in Table 3, generating a key pair and a certificate requests with RSA takes considerably more time than generating a key pair and a certificate requests with *prime256v1*, *secp384r1* or *secp521r1*. These values are used as baseline for the results with PQC and Hybrid-PQC. Note that not *rsa:2048* and *rsa:4096* are not categorized within a SAL. We involved these algorithms in our experiments due to the fact that these algorithms are commonly used.

Table 2: Classical Cryptography certificate generation times (ms).

	SAL	Mean	Median	Std. Deviation	Public Key Size (bytes)	Secret Key Size (bytes)	Signature Size (bytes)
prime256v1	I	20.9	20.5	0.9	64	32	72
rsa:2048	N/A	271.4	238.3	170.1	256	256	256
rsa:3072	I	962.2	820.0	649.9	384	384	384
rsa:4096	N/A	2341.8	2046.3	1502.4	512	512	512
secp384r1	III	28.6	28.3	0.9	96	48	104
secp521r1	V	39.7	39.3	0.9	139	66	132

Post Quantum Cryptography Table 3 gives an overview of the time taken to generate a key pair and a certificate request with PQC. Note that this graph does not make a distinction between the different Security Assurance Levels. What we can derive from these results, is that generating a key pair and certificate requests with PQC does not necessary take more time than generating key pair and certificate requests with classical cryptography.

Table 3: Post-Quantum Cryptography certificate generation times (ms).

	SAL	Mean	Median	Std. Deviation	Public Key Size (bytes)	Secret Key Size (bytes)	Signature Size (bytes)
dilithium2	I	11.8	11.5	0.7	1184	2800	2044
dilithium3	II	12.6	12.2	1.3	1472	3504	2701
dilithium4	III	12.5	12.2	0.9	1760	3856	3366
picnicl1fs	I	19.5	19.4	0.2	33	49	34036
qteslapi	I	18.3	17.7	2.6	14880	5184	2592
qteslapiii	III	62.8	60.7	14.7	38432	12352	5664

Hybrid Post Quantum Cryptography Table 4 shows the results for generating key pairs and certificate requests with Hybrid Post-Quantum Cryptography algorithms. Generating key pairs and certificate requests with these hybrid algorithms take less time than the sum of classical - and Post-Quantum cryptography. This was not what we expected since there are two key's generated, one with the use of classical cryptography and the other with PQC. We found out that generating the key pair with classical cryptography with, e.g. rsa, only uses a single core. Generating the other key pair in a hybrid setup uses any other core. Therefore the overall time is similar to one of the two algorithms which takes the longest time to compute a key pair. Also note that there is no sizes for the public key, secret key or signatures. Since the hybrids are provided by the OpenSSL fork, we could not confidentially say what the exact key - and signature sizes are once standards are defined. We did see that the sizes bigger than the sum of both classical - and PQC algorithms.

Table 4: Hybrid Post-Quantum Cryptography certificate generation times (ms).

	SAL	Mean	Median	Std. Deviation
p256_dilithium2	I	12.3	12.0	0.8
p256_picnicl1fs	I	20.0	20.0	0.2
p256_qteslapi	I	19.1	18.3	3.1
p384_dilithium4	III	20.7	20.4	1.7
p384_qteslapiii	III	70.2	68.4	13.7
rsa:3072_dilithium2	I	867.6	708.0	562.8
rsa:3072_picnicl1fs	I	926.4	812.0	601.9
rsa:3072_qteslapi	I	896.1	760.7	559.5

5.3.2. Signing certificates

Once a certificate requests is generated, it needs to be signed by the Certificate Authority in order to be able to create the certificate to authenticate yourself to a server or client. With our research, we also measured the time it takes to sign certificates generated with different algorithms.

Classical cryptography Table 5 shows the time it takes to sign certificates with classical cryptography. These measurements are used as baseline for a valid comparison with PQC or Hybrid-PQC.

Table 5: Classical Cryptography certificate signing times (ms).

	SAL	Mean	Median	Std. Deviation
prime256v1	I	11.2	10.9	0.6
rsa:2048	N/A	15.3	15.2	0.4
rsa:3072	I	24.3	24.2	0.2
rsa:4096	N/A	40.2	40.1	1.1
secp384r1	III	17.1	17.1	0.2
secp521r1	V	26.4	26.4	0.3

Post Quantum Cryptography Signing certificates with PQC is quite as fast as signing certificates with classical cryptography. Table 6 shows the measured results. Note that the different Security Assurance Levels are all present in the same table.

Table 6: Post-Quantum Cryptography certificate signing times (ms).

	SAL	Mean	Median	Std. Deviation
dilithium2	I	12.1	11.8	0.8
dilithium3	II	13.0	12.5	1.4
dilithium4	III	13.0	12.8	0.9
picnicl1fs	I	27.9	27.9	0.3
qteslapi	I	16.5	15.7	2.6
qteslapiii	III	26.5	24.5	6.2

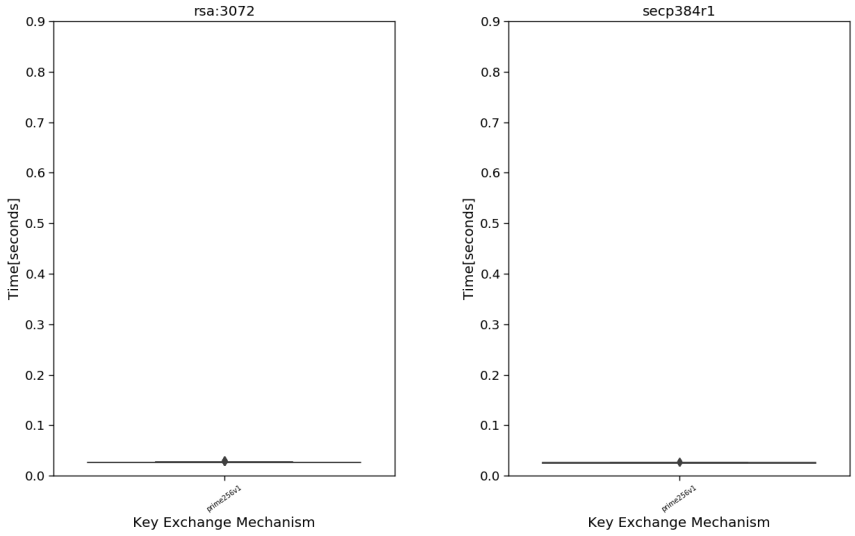
Hybrid Post Quantum Cryptography For the hybrid solution, the results can be seen in Table 7. Just as with generating the key pair, signing the certificates with both classical cryptography and PQC does not take the sum of both algorithms. This is due to the fact that generating a Hybrid-PQC key pair is parallelized across multiple cores. Therefore the overall time is similar the same as the slowest algorithm.

Table 7: Hybrid Post-Quantum Cryptography certificate signing times (ms).

	SAL	Mean	Median	Std. Deviation
p256_dilithium2	I	13.1	12.8	0.7
p256_picnicl1fs	I	28.9	28.9	0.2
p256_qteslapi	I	17.6	16.8	2.4
p384_dilithium4	III	20.3	20.1	0.9
p384_qteslapiii	III	34.2	31.8	6.3
rsa:3072_dilithium2	I	26.5	26.3	0.8
rsa:3072_picnicl1fs	I	42.4	42.3	0.4
rsa:3072_qteslapi	I	31.2	30.2	3.1

5.3.3. Session establishments

Classical cryptography The time required to establish handshakes with classical cryptography was measured in order to form a baseline. See Figure 4 for the results for *rsa:3072* and curve *secp521r1*. As can be seen in these graphs, both *rsa:3072* and *secp521r1* take approximately 26 milliseconds to establish handshakes. For the full set of baseline results please see Appendix A.2.



(a) rsa:3072 (b) secp384r1

Figure 4: Measured handshake times for classical cryptography algorithms

Post Quantum Cryptography For PQC, the time measurements for the handshakes can be seen in Figure 5.

Security Assurance Level 1

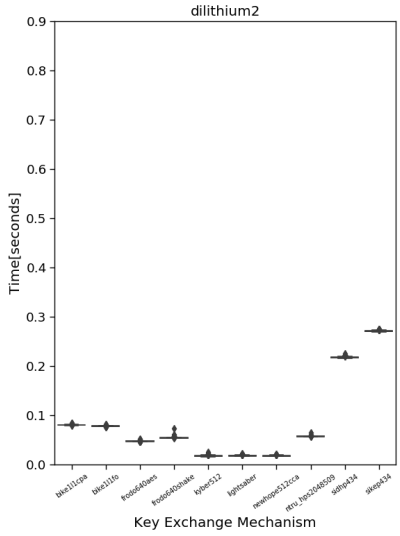
For SAL1 the Digital Signature Algorithm Dilithium2 seems to take the least time for its handshakes. The combination of Dilithium2 with Key Establish Mechanisms *kyber512*, *lightsaber* and *newhope512cca* are the combinations which take about 17 milliseconds to establish a TLS session with mutual authentication. The Digital Signature Algorithm picnicl1fs takes most time for its handshake establishment where also *kyber512*, *lightsaber* and *newhope512cca* seems to be the fastest combinations which take about 44 milliseconds to establish a TLS session with mutual authentication. One other Digital Signature Algorithm is qTeslapi which performs similar to Dilithium2. The performance of qTeslapi can be seen in Appendix A.2.

Security Assurance Level 2

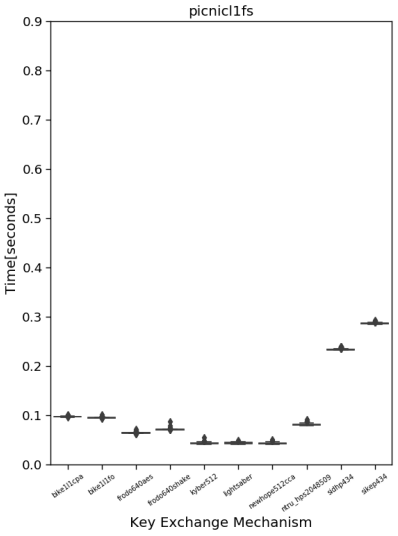
For SAL2 we only had one Digital Signature Algorithm to benchmark, Dilithium3. There were also less possibilities to match Dilithium3 with Key Establishment Algorithms for the mixture to be fully SAL2, namely *sidhp503* and *sikep503*. The combination Dilithium3 and *sidhp503* takes the less time for establishing a TLS session with mutual authentication, as expected. This takes about 325 milliseconds.

Security Assurance Level 3

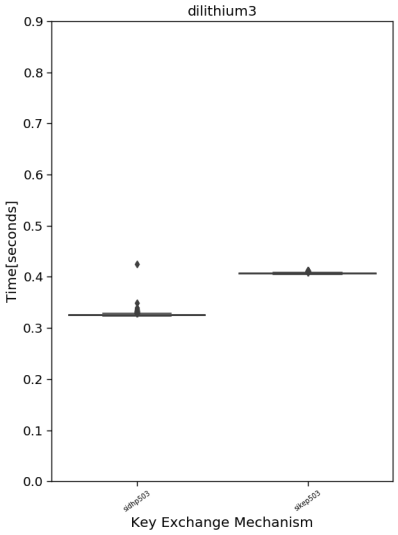
With SAL3 there were two Digital Signature Algorithms, Dilithium4 and qTeslapiii. With both Digital Signature Algotihm, the combination with Key Establishment Algorithm *saber* takes the least time to establish a TLS session with mutual authentication. For Dilithium4 this takes about 19 milliseconds and for qTeslapiii this takes about 33 milliseconds.



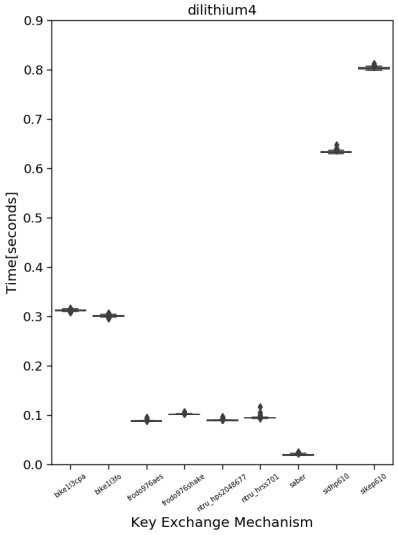
(a) dilithium2



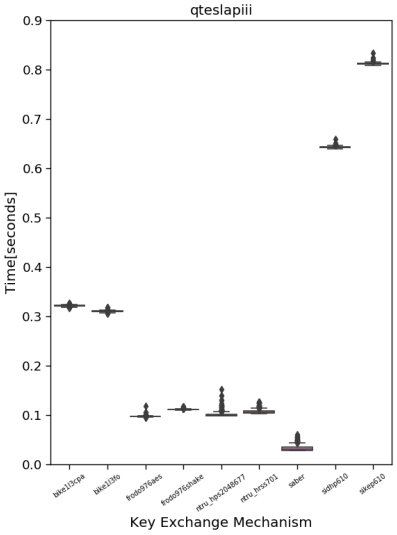
(b) picnicl1fs



(c) dilithium3



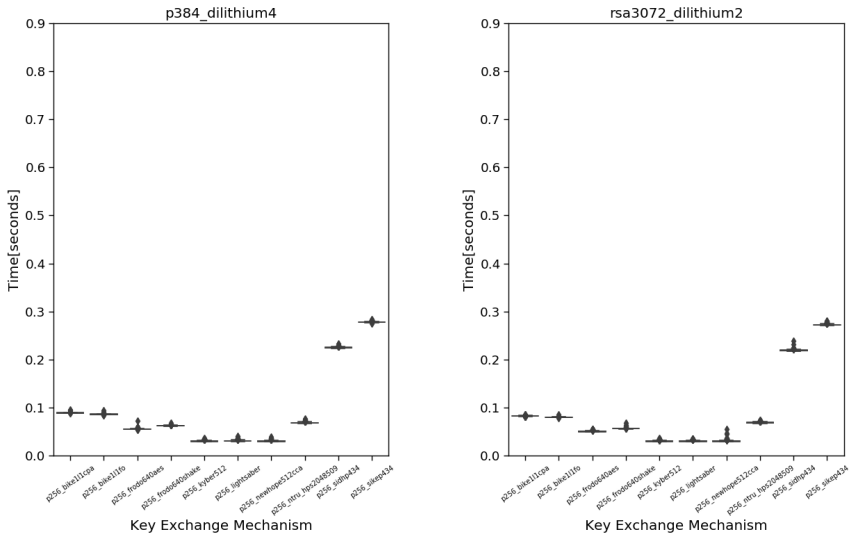
(d) dilithium4



(e) qteslapiii

Figure 5: PQC measurements for handshakes with security assurance levels 1 to 3

Hybrid Post Quantum Cryptography Figure 6 shows the results for two of the hybrid algorithm combinations. For a full overview of all measurements of the hybrid algorithms please see Appendix A.1.2. At this moment fully hybrid algorithm combinations are only available in SAL1, with the exception of *dilithium4* and *qteslapiii*. However, these SAL3 signature algorithms do not have accompanying hybrid key encapsulation algorithms, so SAL1 key encapsulation is used. Differences in key encapsulation is similar to the PQC results with elapsed times below 300 milliseconds. The signature algorithm used does not influence these timing results. *kyber512*, *lightsaber* and *newhope512* seem to be the fastest key encapsulation algorithms in a hybrid setup as well.



(a) p384 & dilithium4 (b) rsa3072 & dilithium2

Figure 6: Measured handshake times for hybrid post quantum cryptography

5.4. Practical feasibility of transitioning to post-quantum cryptography

With the results described above, we can give an overview of the practical feasibility of transitioning to post-quantum cryptography.

5.4.1. Generating key pairs & certificates

Post Quantum Cryptography Compared to the baseline set with traditional signature schemes the PQC alternatives perform very well. The generation of key pairs and accompanying certificates is bound to tens of milliseconds and can therefore easily be used in 'on-demand' situations. Spinning up new microservices on a large scale and locally generating the required key pair and certificates thus are of no concern. Higher security levels, such as *dilithium4* and *qteslapiii*, do also have results within similar bounds resulting the transition to PQC signature feasible in terms of elapsed time. However, in the case of pre-generated key pairs and certificates one should consider the sizes of the certificates and keys. Although well within reasonable numbers, the qTesla family produces the largest key pairs and signatures, about two times the size of the Dilithium variants. The public key, the private key and signature are 15 KB, 5 KB and 2.5 KB respectively [22]. Picnic has negligible size of the keys, though the signature consists of about 35 KB. Still, these numbers do not imply any complications in a large scale microservice environment. The only case where this might be a problem is when there are severe hardware constraints such as in embedded devices.

Hybrid Post Quantum Cryptography Hybrid signature produce similar results in comparison to the PQC alternatives. However, RSA based hybrid signature schemes do have a significant increase in computation time of about a factor 10. The results still show the elapsed times to be under a second, but for SAL 1 security we consider this to be a long time. On demand generation of such hybrid schemes may have somewhat of an impact on the stability of large scale systems with high fluctuations in demand. Hybrid signature schemes are valuable when transitioning to PQC in different parts of the architecture. Thus, the time that less efficient hybrid certificates are used is limited to the time that is needed to complete the transition. The design of PQC in TLS is still being standardized. More efficient communications schemes using separate certificates may be used in the future [28], which will produce different performance results.

5.4.2. Signing certificates

Post Quantum Cryptography The PQC signature results are promising, often equally as fast and in some cases even outperforming traditional signature schemes. Signature sizes are also of comparable size with the exception of the relatively large signatures of the *picnicl1fs* signature scheme at about 35 KB. As these PQC results are so close to the traditional signature schemes, we expect there to be no large barriers to transitioning to PQC in terms of computation time. The speed at which signatures are created and verified will most probably not be the bottleneck in the signing and verification process. If no certificate stores are locally available, network latency will most likely be more significant in the elapsed time. Again, only in highly constrained environments problems may occur. For example, with the larger signatures of *picnicl1fs* that might not fit in the device's memory.

Hybrid Post Quantum Cryptography As with the PQC signing results, the hybrid signature schemes produce similar results as compared to the sum of traditional and PQC schemes. With this come the same conclusions: process time of the signing process will not be a significant bottleneck to the applications latency. However, hybrid signatures consist of both a traditional and PQC scheme. This means that the signature sizes are larger, and when both certificates are required to be verified, the verification process will take longer.

5.4.3. Session establishments

Post Quantum Cryptography As we can see from the results the elapsed time each handshake takes is strongly dependant on the PQC KEM that is used. Some of these PQC KEMs are much less efficient compared to the available alternatives. In addition, the SAL variants of certain KEMs also have a large impact on the elapsed time. However, most PQC KEMs keep the handshike time to less than half a second. The isogeny-based PQC algorithms are by far performing the worst, compared to the other KEMs. Handshakes of a full second has a large impact on the availability of the system, especially in large scale environments where a large number of short sessions are required. For example, if a web page is requested from such an environment, one whole second for the setting up of a secured connection is a long time. Especially at higher SALs, the time of isogeny based PQC goes beyond the one second mark.

Lattice based PQC algorithms *Kyber*, *Saber* and *Frodo* variants are consistently below or around 100 milliseconds. In combination with their relatively small key sizes these algorithms make good candidates for usage in large scale environments where efficiency is required. Also in restricted environments with limited resources these algorithms can operate accordingly. The *Bike* family performs well at a similar elapsed time compared to the lattice based algorithms previously mentioned. However, at SAL 3 performance seems to drop at around 300 milliseconds.

Hybrid Post Quantum Cryptography The performance results of the hybrid PQC algorithms are similar to the PQC algorithms. This is due to the fact that a single KEM is specified during client handshake. So, in this case hybrid PQC setups can be used to transition to PQC only algorithms without any loss on the Key exchange. However, the hybrid signature does includes both a non-PQC and a PQC signature. In this case, extra data is traversing the connection between the client and the server. Some of these signatures are of considerable size compared to traditional signature schemes, and on connections with limited bandwidth this make take some extra time. In a microservice environment this should not be of much concern, unless there are very specific restrictions that limits bandwidth.

6. Discussion

6.1. NIST PQC Standardization Process

The NIST standardization process is still ongoing. Therefore it is possible that algorithms, which are tested in this paper, will be modified or even are considered insecure in the future. With modification, it is possible that our result do not comply with the newly modified algorithm.

6.2. Test Setup

There are components in our test setup which impacted our results. These components are discussed below.

6.2.1. Optimal situation

Our test setup is considered as an optimal situation. This is due to the fact that both TLS Server and TLS Client are located in the same server-rack and both hosts are part of the same layer 3 domain. This means that there is, other than a network switch, no network equipment involved. Also we did not emulate an environment where several hundreds or thousands TLS Clients. Once the TLS Server and Clients are located further apart, the Round Trip Time will increase and this will impact the time it takes to establish a TLS Session. Since we did not emulate hundreds or thousands of TLS Clients, we did not test the behaviour of the TLS Server with such workload.

6.2.2. Algorithm optimizations

With conducting our tests, we did not optimize the algorithms. Our tests are based on the 'out-of-the-box' experience of the algorithms. Therefore it is possible that results may vary when algorithm optimization is applied. In addition, we recognize the improvement of these algorithms in the future. Especially when the NIST has determined the new standard we expect more hardware based implementation to emerge and thus significantly improve performance for all categories of PQC.

6.2.3. Certificate generation

The time it takes to generate and sign a certificate is measured and part of the conclusion. With this approach, we tested an 'on-demand' scenario. It is possible to pre-generate and sign certificates. Therefore, once a new microservice is needed, the certificate is already generated and signed and thus ready for use. With this approach, both certificate generation and signing is excluded from the time it takes to create a new microservice. Also, the results for hybrid signature generation show that hybrid signature schemes are similar to their non-hybrid variants in terms of performance. We suspect that this is the case due to parallelization of the separate algorithms, thus the total elapsed time will be similar to the signature scheme taking the most time.

7. Conclusion

During this research we found out that using the OpenSSL fork with third-party applications was not as simple as using the original OpenSSL library. The main problem is that development is ungoing and the API does not include calls to PQC based TLS. To some degree, this is due to the lack of standardized cryptography suites in the OpenSSL library API. Discussions on how to best specify the PQC algorithms in TLS 1.3 are still ongoing. Therefore we consider the forked OpenSSL library at the moment of this writing less practical for implementations in microservice environments. However, this is bound to change with the development of a third party library API.

The results show that duration of a full TLS 1.3 handshake is dependant on the KEM as well as what security level variant is used. Generally we see lattice-based PQC algorithms with good performance at below or around 100 milliseconds. Code-based PQC is shown to perform slightly worse at higher security levels. Isogeny-based PQC algorithms are significantly slower than its alternatives. Most PQC and hybrid KEMs on the measured security levels 1 and 3 perform well enough to be used in large scale microservice environments, at around 100 millisecond per handshake. The main exception are the isogeny-based PQC algorithms which have significantly larger handshake times compared to the alternatives and can take whole seconds per handshake at higher security levels.

Key pair, certificate signing and signature generation are considered similar in terms of computation time. We find that all signature algorithms are usable in large scale microservice environments in regards of computation time. The PQC signature schemes vary in key and certificate sizes which can be an impact on the system depending on the microservice environment and its restrictions. Hybrid signature schemes are comparable in computation time, however as both classical and PQC signatures are embedded into the hybrid signature, the size of signatures does increase. Depending on what PQC signature scheme is used this may have an impact on restricted environments in terms of bandwidth or available memory. In large scale microservice environments this will not have a significant impact.

8. Future Work

8.1. Post Quantum Cryptography in Java

As at this moment there is significant development time needed to create a bridge between the Java environment and the OpenSSL library. It will be interesting to see how the development of a general interface for providers running on different software stacks can be implemented. Also, the standardization of the PQC TLS suites should bring clarity in developing TLS interfaces for all software stacks. That way no explicit conversion of cipher suites or OIDs is needed in the identification process for different software stacks.

8.2. Further Testing on PQC Algorithms

Since not all algorithms are implemented in the OpenSSL fork, future work could be to test newly implemented algorithms. This way, there are more possibilities to create Security Assurance Level 3 and 5 only combinations. Also, the adaptation of the OpenSSL TLS interface to include different PQC algorithms to be used by third party libraries will be interesting for future performance research.

Because the used OpenSSL fork mainly contains algorithms in NIST security categories 1 to 3, these will be the main focus for the results. However, it is possible to create combinations with Key Exchange Mechanisms and Digital Signature Algorithms of different levels. Therefore it would be possible to assure a level V Key Exchange Mechanism with a lower level Digital Signature Algorithm.

Performance tests in a representative microservice environment will also be interesting. Different traffic profiles may be used to simulate such an environment. In addition, networking differences may be taken into account. For example a multitude of latency and throughput restrictions could be applied to the connections between the microservices and the message bus.

References

[1] W. Buchanan and A. Woodward, "Will quantum computers be the end of public key encryption?" *Journal of Cyber Security Technology*, vol. 1, no. 1, pp. 1–22, 2017.

[2] J. Kreps, N. Narkhede, J. Rao *et al.*, "Kafka: A distributed messaging system for log processing," in *Proceedings of the NetDB*, 2011, pp. 1–7.

[3] G. Alagic, G. Alagic, J. Alperin-Sheriff, D. Apon, D. Cooper, Q. Dang, Y.-K. Liu, C. Miller, D. Moody, R. Peralta *et al.*, *Status report on the first round of the NIST post-quantum cryptography standardization process*. US Department of Commerce, National Institute of Standards and Technology, 2019.

[4] Computer Security Resource Center, "Round 2 submissions - post-quantum cryptography — csrc," https://csrc.nist.gov/Projects/post-quantum-cryptography/round-2-submissions, accessed on January 8th 2020.

[5] D. Stebila and M. Mosca, "Open quantum safe," https://openquantumsafe.org/, accessed on January 8th 2020.

[6] E. Crockett, C. Paquin, and D. Stebila, "Prototyping post-quantum and hybrid key exchange and authentication in tls and ssh," in *NIST 2nd Post-Quantum Cryptography Standardization Conference 2019*, 2019.

[7] D. Stebila and M. Mosca, "Oqs-openssl 1.1.1," https://github.com/open-quantum-safe/openssl/tree/OQS-OpenSSL_1_1_1-stable, accessed on January 9th 2020.

[8] D. Stebila, J. Rijneveld, M. J. Kannwhischer, and T. Wiggers, "Pqclean," https://github.com/PQClean/PQClean, accessed on February 5th 2020.

[9] H. Nejatollahi, N. Dutt, S. Ray, F. Regazzoni, I. Banerjee, and R. Cammarota, "Post-quantum lattice-based cryptography implementations: A survey," *ACM Computing Surveys (CSUR)*, vol. 51, no. 6, pp. 1–41, 2019.

[10] R. J. McEliece, "A public-key cryptosystem based on algebraic," *Coding Thv*, vol. 4244, pp. 114–116, 1978.

[11] V.-F. Dragoi, T. Richmond, D. Bucerzan, and A. Legay, "Survey on cryptanalysis of code-based cryptography: From theoretical to physical attacks," 05 2018, pp. 215–223.

[12] D. J. Bernstein and T. Lange, "ebacs: Ecrypt benchmarking of cryptographic systems," https://bench.cr.yp.to, accessed 5 Febuari 2020.

[13] J.-C. Faugère, K. Horan, D. Kahrobaei, M. Kaplan, E. Kashefi, and L. Perret, "Fast quantum algorithm for solving multivariate quadratic equations," *arXiv preprint arXiv:1712.07211*, 2017.

[14] D. Jao and L. De Feo, "Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies," in *International Workshop on Post-Quantum Cryptography*. Springer, 2011, pp. 19–34.

[15] D. J. Bernstein, A. Hülsing, S. Kölbl, R. Niederhagen, J. Rijneveld, and P. Schwabe, "The sphincs+ signature framework," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019, pp. 2129–2146.

[16] E. Rescorla, "Keying material exporters for transport layer security (tls)," Internet Requests for Comments, RFC Editor, RFC 8446, 03 2010. [Online]. Available: https://www.rfc-editor.org/info/rfc5705

[17] ——, "The transport layer security (tls) protocol version 1.3," Internet Requests for Comments, RFC Editor, RFC 8446, 09 2018. [Online]. Available: https://www.rfc-editor.org/info/rfc8446

[18] P. Dobbelaere and K. S. Esmaili, "Kafka versus rabbitmq: A comparative study of two industry reference publish/subscribe implementations: Industry paper," in *Proceedings of the 11th ACM International Conference on Distributed and Event-Based Systems*, ser. DEBS '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 227–238. [Online]. Available: https://doi.org/10.1145/3093742.3093908

[19] Facebook, "Scribe," http://github.com/facebook/scribe, accessed on February 7th 2020.

[20] Apache Software Fountdation, "Kafka documentation," http://kafka.apache.org/documentation/, accessed on January 8th 2020.

[21] Oracle, "Jdk providers documentation," https://docs.oracle.com/en/java/javase/11/security/, accessed on January 12th 2020.

[22] D. Stebila and M. Mosca, "Oqs-liboqs," https://github.com/open-quantum-safe/liboqs, accessed on January 10th 2020.

[23] The Legion of the Bouncy Castle, "The legion of the bouncy castle homepage," http://www.bouncycastle.org/, accessed on January 22th 2020.

[24] D. Peter, "Hyperfine - command line benchmarking tool," https://github.com/sharkdp/hyperfine, accessed on January 21th 2020.

[25] M. Wolff, "Heaptrack - a heap memory profiler for linux," https://github.com/KDE/heaptrack, accessed on January 22th 2020.

[26] The OpenSSL Project, "Openssl 1.1.1 official documentation," https://www.openssl.org/docs/man1.1.1/man1/, accessed on February 5th 2020.

[27] S. L. Douglas, "Wildfly-openssl 1.0.10.final," https://github.com/wildfly-security/wildfly-openssl, accessed on January 17th 2020.

[28] D. Steblia, S. Fluhrer, and S. Gueron, "Design issues for hybrid key exchange in tls 1.3," https://tools.ietf.org/id/draft-stebila-tls-hybrid-design-01.html#rfc.section.3.1.1, internet draft. Work in progress.

A. Appendixes

A.1. PQC algorithms available in the OpenSSL Fork

A.1.1. Post-Quantum Cryptography Algorithms

The table below gives an overview of the available (as of this writing) Post-Quantum Cryptography algorithms with the OpenSSL project by Open Quantum Safe [7].

Table 8: Post-Quantum Cryptography algorithms available in the OpenSSL Fork

Level	**Post Quantum Key Exchange Mechanisms**	**Post Quantum Digital Signature Algorithms**
I	bike1l1cpa, bike1l1fo, frodo640aes, frodo640shake, kyber512, newhope512cca, ntru_hps2048509, lightsaber, sidhp434, sikep434	dilithium2 picnicl1fs qteslapi
II	sidhp503, sikep503	dilithium3
III	bike1l3cpa, bike1l3fo, frodo976aes, frodo976shake, ntru_hps2048677, ntru_hrss701, saber, sidhp610, sikep610, sidhp751, sikep751	dilithium4 qteslapiii
IV	None	None
V	frodo1344aes, frodo1344shake, kyber1024, newhope1024cca, ntru_hps4096821, firesaber	None

A.1.2. Hybrid Post-Quantum Cryptography Algorithms

The table below gives an overview of the available (as of this writing) Hybrid Post-Quantum Cryptography algorithms with the OpenSSL Fork delivered by Open Quantum Safe [7].

Table 9: Hybrid Post-Quantum Cryptography algorithms available in the OpenSSL Fork

Level	**Hybrid Post Quantum Key Exchange Mechanisms**	**Hybrid Post Quantum Digital Signature Algorithms**
I	p256_bike1l1cpa, p256_bike1l1fo, p256_frodo640aes, p256_frodo640shake, p256_kyber512, p256_newhope512cca, p256_ntru_hps2048509, p256_lightsaber, p256_sidhp434, p256_sikep434.	rsa3072_dilithium2, p256_dilithium2, rsa3072_picnicl1fs, p256_picnicl1fs, rsa3072_qteslapi, p256_qteslapi
II	None	None
III	None	p384_dilithium4, p384_qteslapiii
IV	None	None
V	None	None

A.2. Results

This part of the appendix gives the full results of our experiments. We did not include all the results into the main text since this would decrease the readability of the text.

A.2.1. Classical Cryptography Algorithms

Certificate generation Table 10 is the same table as showed in paragraph 5.3.1. For a visual comparison, Figure 7 is diplayed below.

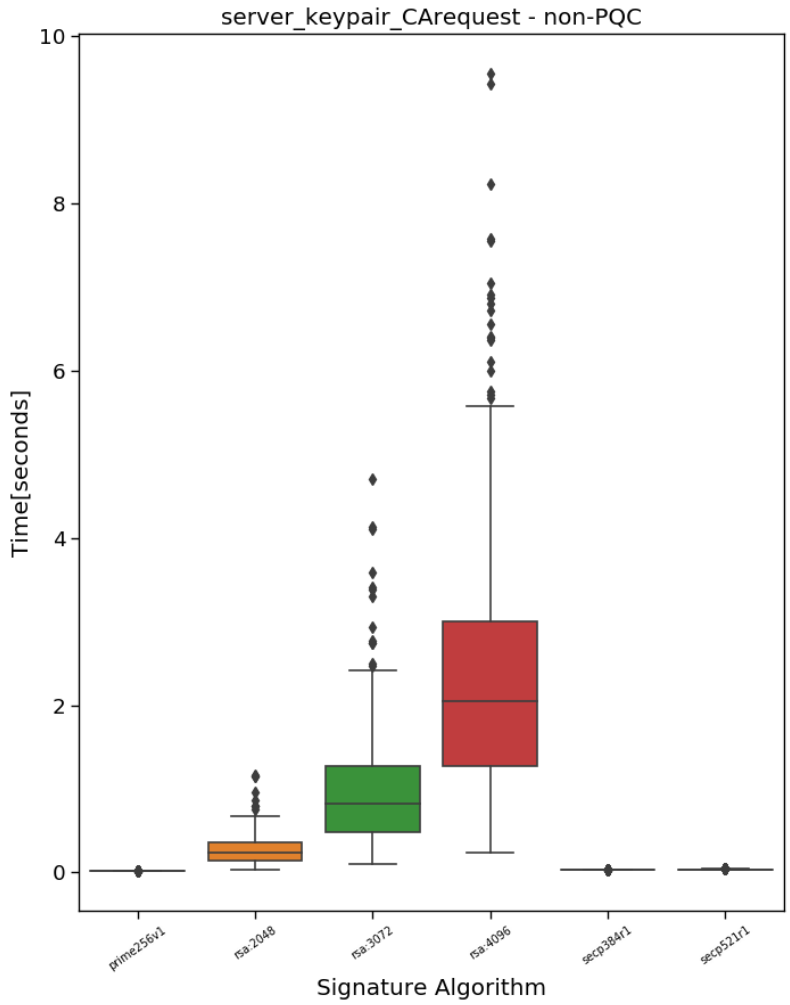


Figure 7: Generating certificates with classical cryptography

Table 10: Classical Cryptography certificate generation times (ms).

	SAL	Mean	Median	Std. Deviation	Public Key Size (bytes)	Secret Key Size (bytes)	Signature Size (bytes)
prime256v1	I	20.9	20.5	0.9	64	32	72
rsa:2048	N/A	271.4	238.3	170.1	256	256	256
rsa:3072	I	962.2	820.0	649.9	384	384	384
rsa:4096	N/A	2341.8	2046.3	1502.4	512	512	512
secp384r1	III	28.6	28.3	0.9	96	48	104
secp521r1	V	39.7	39.3	0.9	139	66	132

Certificate signing Table 11 is the same table as showd in paragraph 5.3.2. Figure 8 is shown to give a visual overview of the values shown in the table.

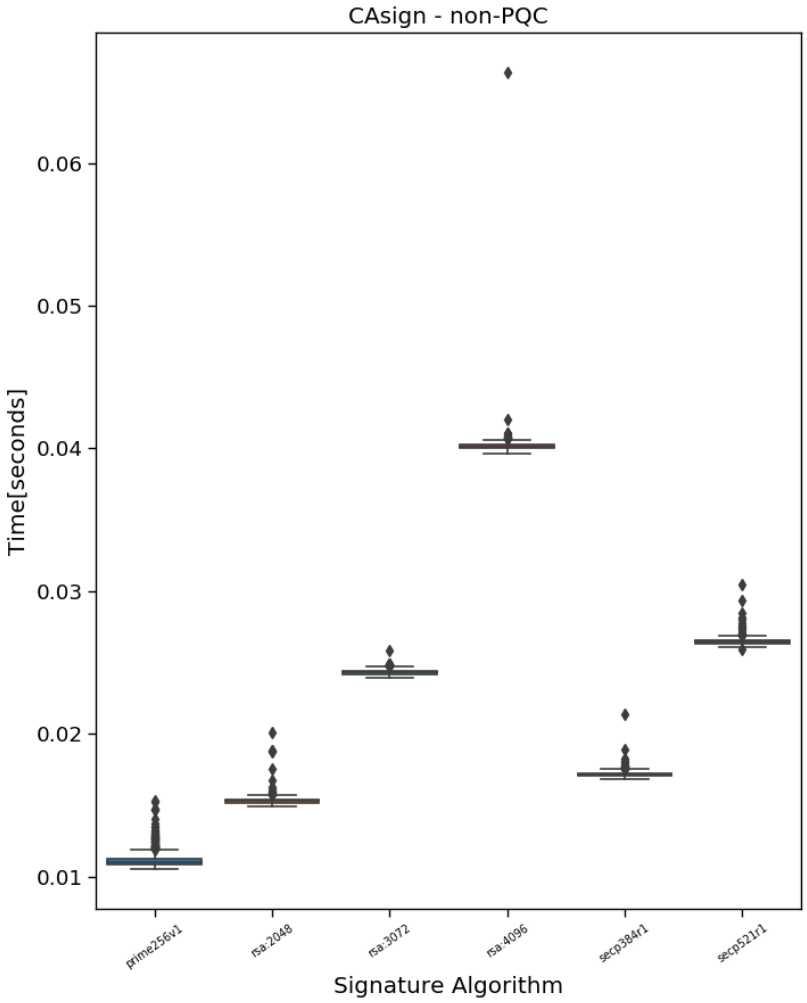
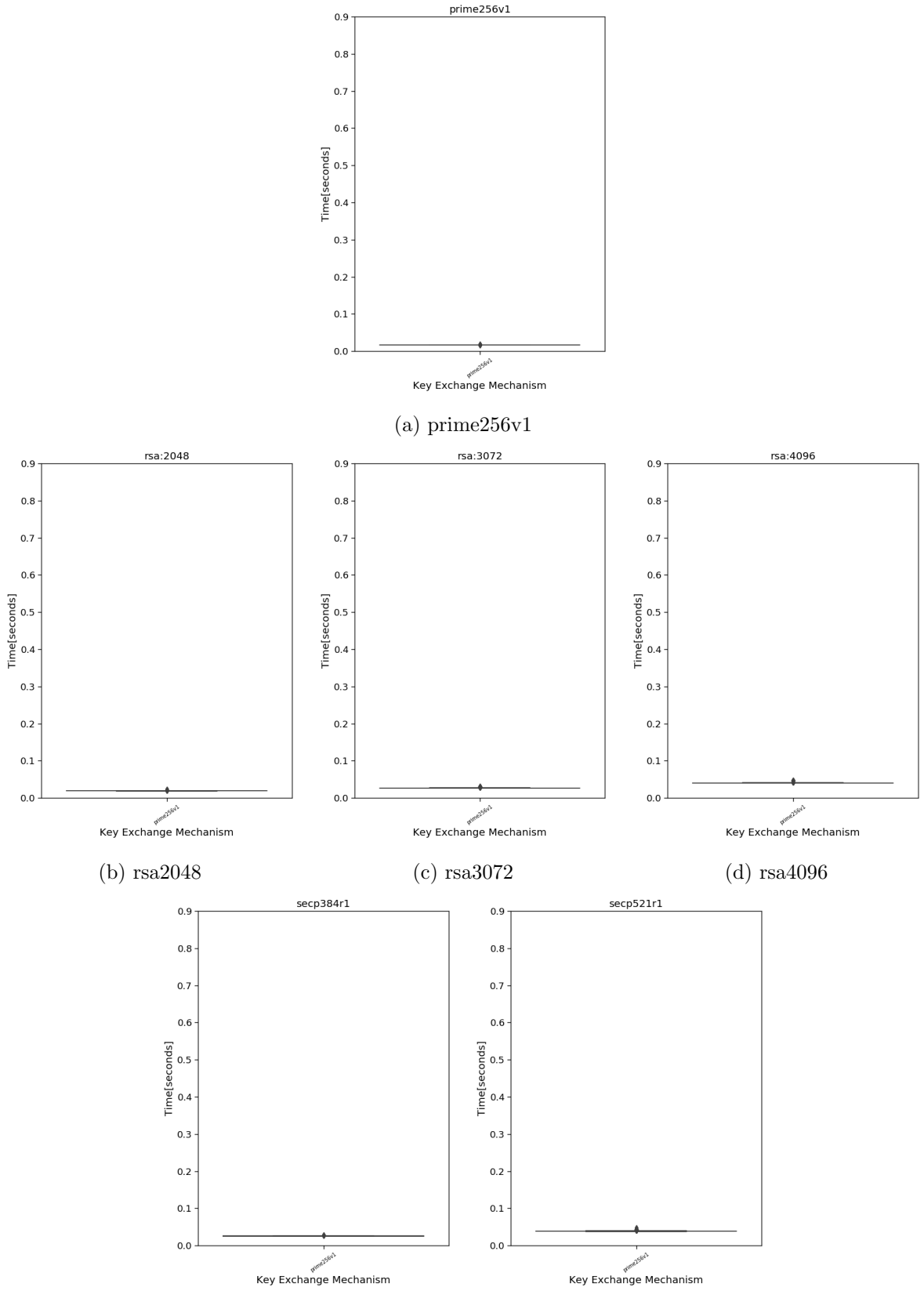


Figure 8: Time measurements for generating signing using classical cryptography algorithms.

Table 11: Classical Cryptography certificate signing times (ms).

	Mean	**Median**	**Std. Deviation**
prime256v1	11.2	10.9	0.6
rsa:2048	15.3	15.2	0.4
rsa:3072	24.3	24.2	0.2
rsa:4096	40.2	40.1	1.1
secp384r1	17.1	17.1	0.2
secp521r1	26.4	26.4	0.3

Handshake times This section gives an overview of the handshake-times for the classical cryptography algorithms. For prime256v1, see Figure 9a. For RSA, see Figures 9b, 9c and 9d. For secp384r1 9e and for secp521r1, see Figure 9f. Table 12 gives an overview of the measured values.



(a) prime256v1

(b) rsa2048

(c) rsa3072

(d) rsa4096

(e) secp384r1

(f) secp521r1

Figure 9: Full measurements of classical cryptography handshakes

Table 12: Classical Cryptography handshake times (ms).

	Mean	Median	Std. Deviation
prime256v1	16.7	16.7	0.2
rsa:2048	19.3	19.2	0.4
rsa:3072	26.9	26.7	0.7
rsa:4096	40.9	40.7	1.1
secp384r1	25.8	25.7	0.3
secp521r1	39.2	38.8	1.1

A.2.2. Post-Quantum Cryptography Algorithms

Certificate generation Table 13 gives an overview of the measured time it takes to generate certificates with PQC algorithms. Figure 10 gives an visual overview of these values.

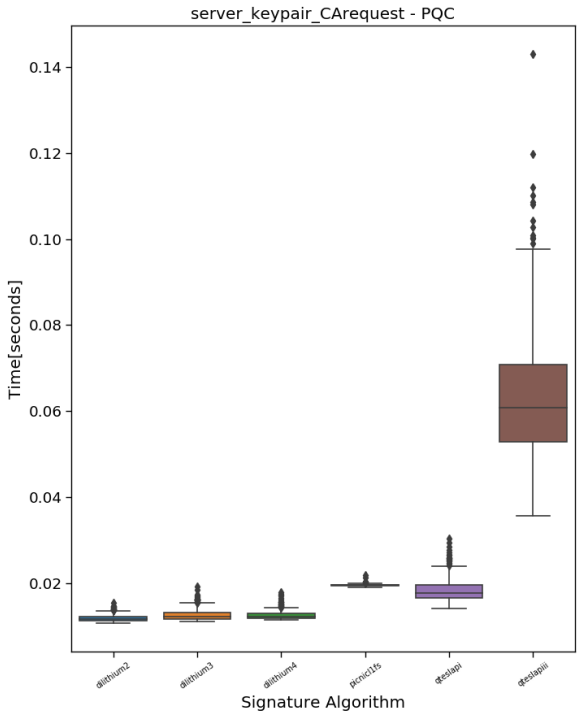


Figure 10: Time measurements for generating certificates using PQC algorithms.

Table 13: Post-Quantum Cryptography certificate generation times (ms).

	SAL	Mean	Median	Std. Deviation	Public Key Size (bytes)	Secret Key Size (bytes)	Signature Size (bytes)
dilithium2	I	11.8	11.5	0.7	1184	2800	2044
dilithium3	II	12.6	12.2	1.3	1472	3504	2701
dilithium4	III	12.5	12.2	0.9	1760	3856	3366
picnicl1fs	I	19.5	19.4	0.2	33	49	34036
qteslapi	I	18.3	17.7	2.6	14880	5184	2592
qteslapiii	III	62.8	60.7	14.7	38432	12352	5664

Certificate signing For the time measured for certificate signing an overview can be found in Table 14. We give a visual overview of these values in Figure 11.

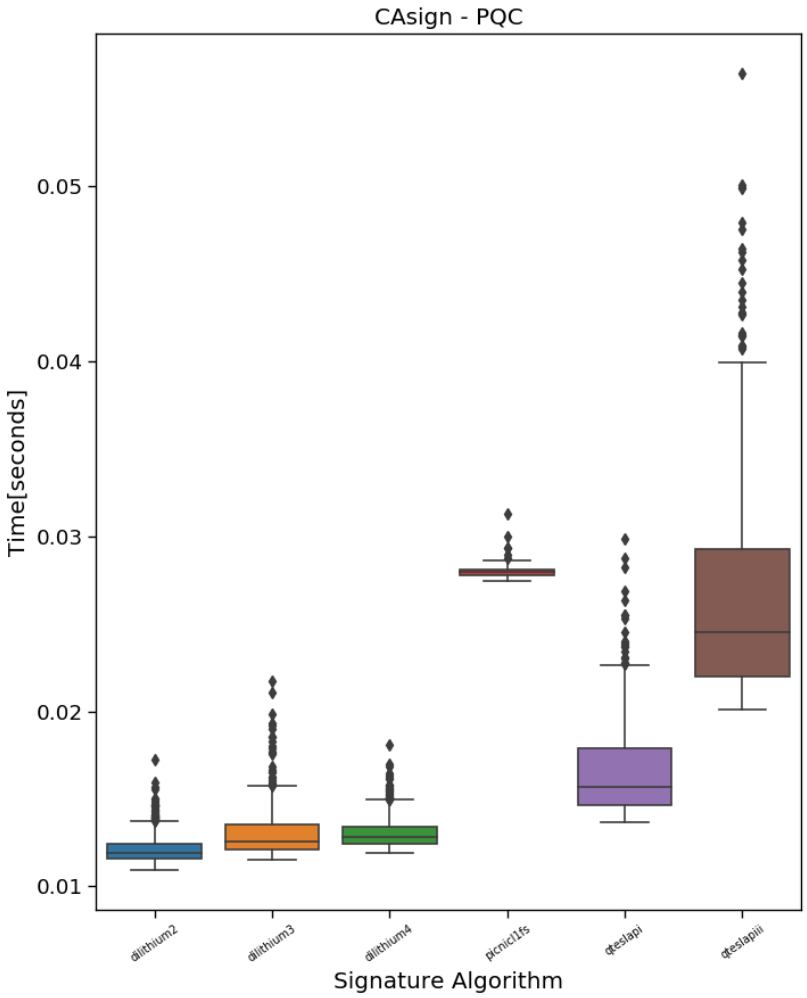


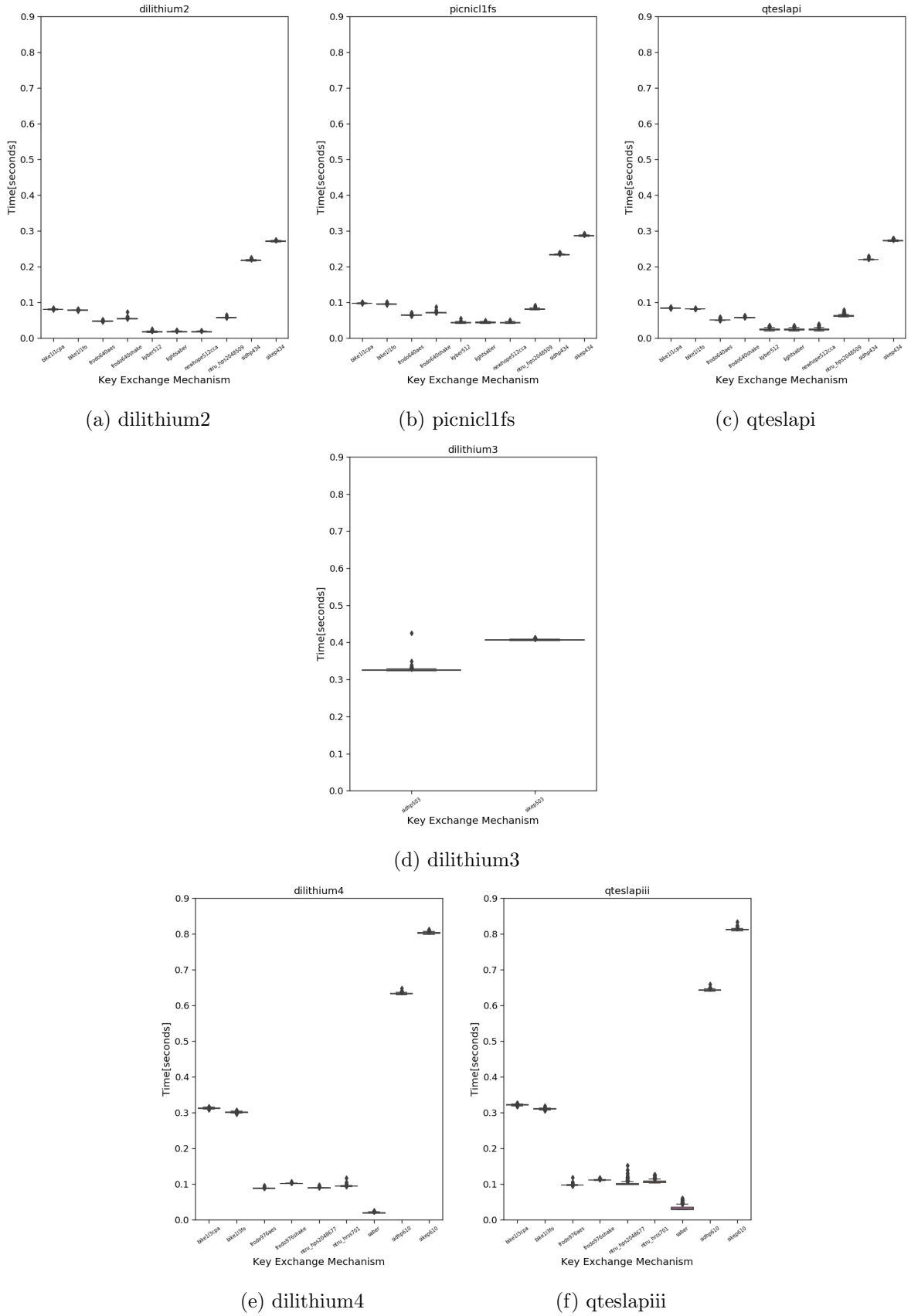
Figure 11: Time measurements for generating signing PQC algorithms.

Table 14: Post-Quantum Cryptography certificate signing times (ms).

	SAL	**Mean**	**Median**	**Std. Deviation**
dilithium2	I	12.1	11.8	0.8
dilithium3	II	13.0	12.5	1.4
dilithium4	III	13.0	12.8	0.9
picnicl1fs	I	27.9	27.9	0.3
qteslapi	I	16.5	15.7	2.6
qteslapiii	III	26.5	24.5	6.2

Handshake times The time taken for establishing a TLS handshake with PQC algorithms is shown in Tables 15 up to 20. For each signature algorithm there is a table with its combinations. A visual overview is given in Figure 12.

Level 1 algorithms:



(a) dilithium2 (b) picnicl1fs (c) qteslapi

(d) dilithium3

(e) dilithium4 (f) qteslapiii

Figure 12: PQC measurements for handshakes with security assurance levels 1 to 3

Table 15: dilithium2 handshake times (ms).

dilithium2 with:	**Mean**	**Median**	**Std. Deviation**
bike1l1cpa	80.3	80.3	0.5
bike1l1fo	78.0	78.0	0.5
frodo640aes	47.4	47.4	0.4
frodo640shake	54.3	54.1	1.0
kyber512	17.6	17.4	0.9
lightsaber	17.8	17.7	0.8
newhope512cca	17.7	17.6	0.7
ntru_hps2048509	57.2	057.1	0.5
sidhp434	217.6	217.4	1.2
sikep434	270.7	270.5	1.0

Table 16: picnicl1fs handshake times (ms).

picnicl1fs with:	**Mean**	**Median**	**Std. Deviation**
bike1l1cpa	97.5	97.5	0.5
bike1l1fo	95.0	94.9	0.6
frodo640aes	64.4	64.3	0.6
frodo640shake	71.1	71.0	0.9
kyber512	43.6	44.0	1.2
lightsaber	43.6	44.1	1.0
newhope512cca	43.3	42.8	1.1
ntru_hps2048509	81.1	80.6	1.3
sidhp434	233.6	233.4	1.1
sikep434	286.2	286.1	0.9

Table 17: qteslapi handshake times (ms).

qteslapi with:	**Mean**	**Median**	**Std. Deviation**
bike1l1cpa	83.8	83.7	0.5
bike1l1fo	81.3	81.3	0.4
frodo640aes	50.4	50.3	0.9
frodo640shake	57.2	57.2	0.5
kyber512	24.1	23.4	2.7
lightsaber	23.8	23.0	2.7
newhope512cca	24.3	23.4	2.9
ntru_hps2048509	62.4	61.5	2.7
sidhp434	219.9	219.8	1.0
sikep434	272.7	272.5	1.1

Level 2 algorithms:

Table 18: diltihium3 handshake times (ms).

dilithium3 with:	**Mean**	**Median**	**Std. Deviation**
sidhp503	326.1	325.3	5.0
sikep503	406.5	406.3	1.2

Level 3 algorithms:

Table 19: dilithium4 handshake times (ms).

dilithium4 with:	**Mean**	**Median**	**Std. Deviation**
bike1l3cpa	312.2	312.3	1.0
bike1l3fo	300.9	300.9	1.1
frodo976aes	87.8	87.7	0.6
frodo976shake	101.5	101.5	0.4
ntru_hps2048677	89.4	89.3	0.6
ntru_hrss701	94.4	94.3	1.2
saber	19.6	19.4	1.0
sidhp610	632.9	632.6	1.8
sikep610	802.6	802.5	1.6

Table 20: qteslapiii handshake times (ms).

qteslapiii with:	**Mean**	**Median**	**Std. Deviation**
bike1l3cpa	321.4	321.3	1.2
bike1l3fo	310.2	310.3	1.2
frodo976aes	97.0	96.8	1.3
frodo976shake	111.0	110.9	0.8
ntru_hps2048677	101.5	099.1	5.8
ntru_hrss701	106.4	104.3	4.3
saber	32.9	30.3	6.1
sidhp610	642.6	642.3	1.7
sikep610	812.3	812.1	1.9

A.2.3. Hybrid Cryptography Algorithms

Certificate generation Figure 13 gives a visual overview of the time it takes to generate a keypair and certificate. This is a representation fo the values shown in Table 21.

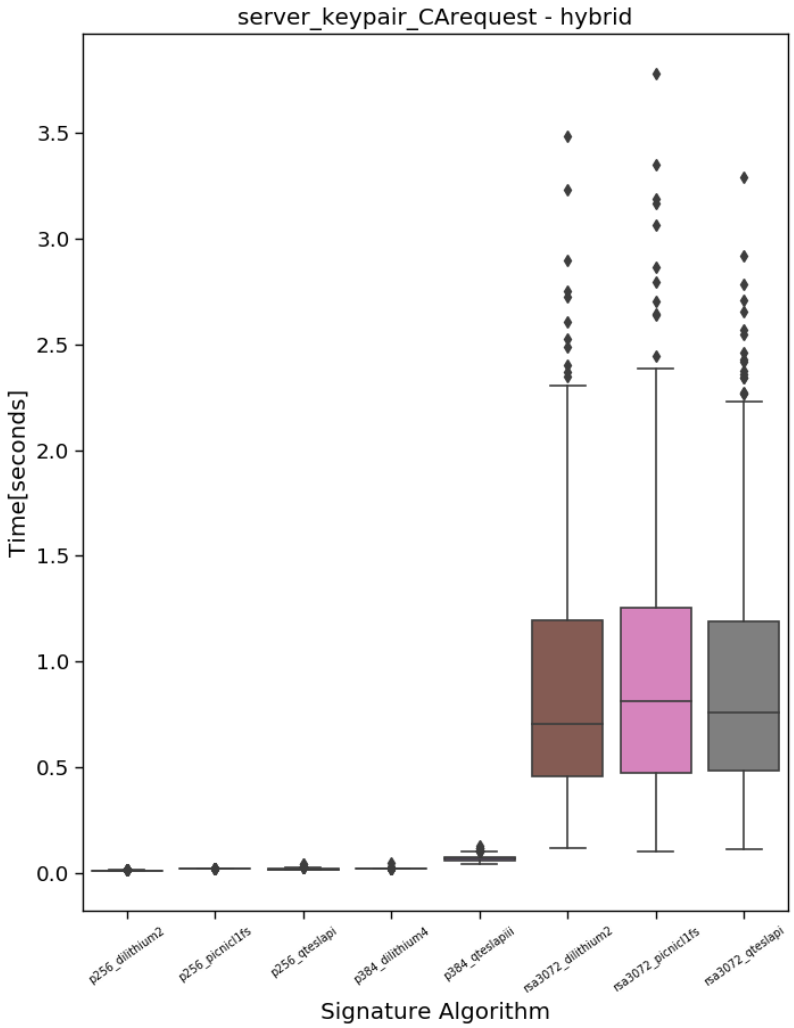


Figure 13: Time measurements for generating certificates using Hybrid PQC algorithms.

Table 21: Hybrid Post-Quantum Cryptography certificate generation times (ms).

	SAL	Mean	Median	Std. Deviation
p256_dilithium2	I	12.3	12.0	0.8
p256_picnicl1fs	I	20.0	20.0	0.2
p256_qteslapi	I	19.1	18.3	3.1
p384_dilithium4	III	20.7	20.4	1.7
p384_qteslapiii	III	70.2	68.4	13.7
rsa:3072_dilithium2	I	867.6	708.0	562.8
rsa:3072_picnicl1fs	I	926.4	812.0	601.9
rsa:3072_qteslapi	I	896.1	760.7	559.5

Certificate signing The time it takes to sign certificates with Hybrid-PQC algorithms, is shown in Table 22. Figure 14 gives a visual overview of these values.

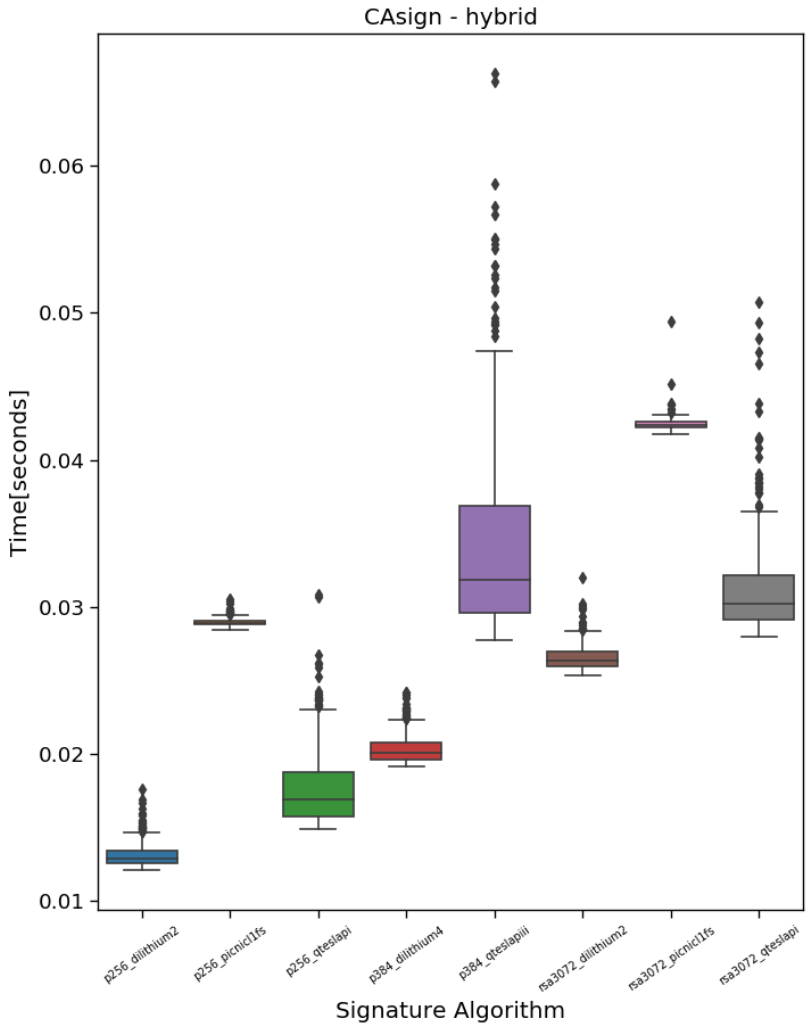
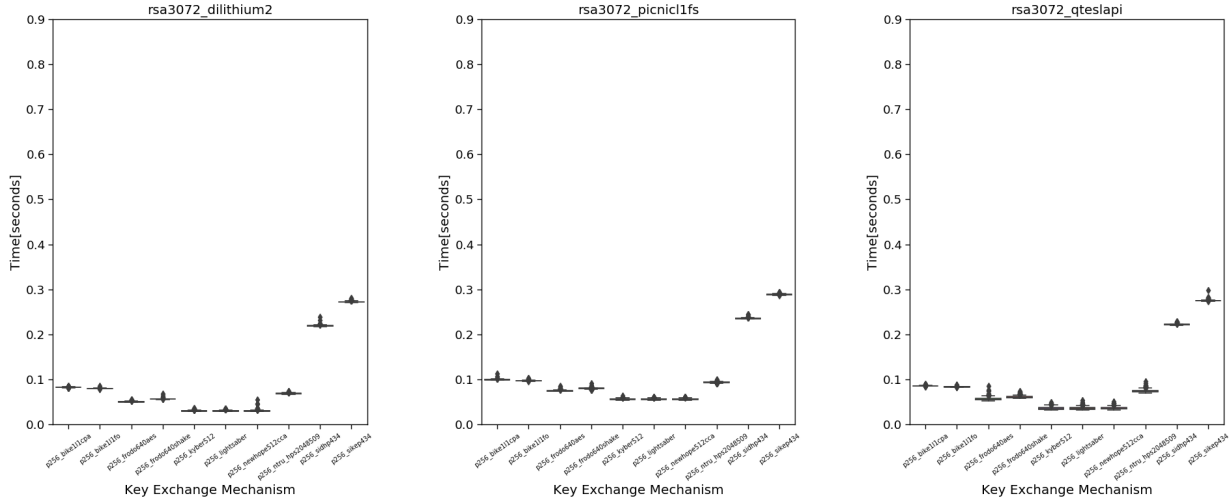


Figure 14: Time measurements for generating signing Hybrid PQC algorithms.

Table 22: Hybrid Post-Quantum Cryptography certificate signing times (ms).

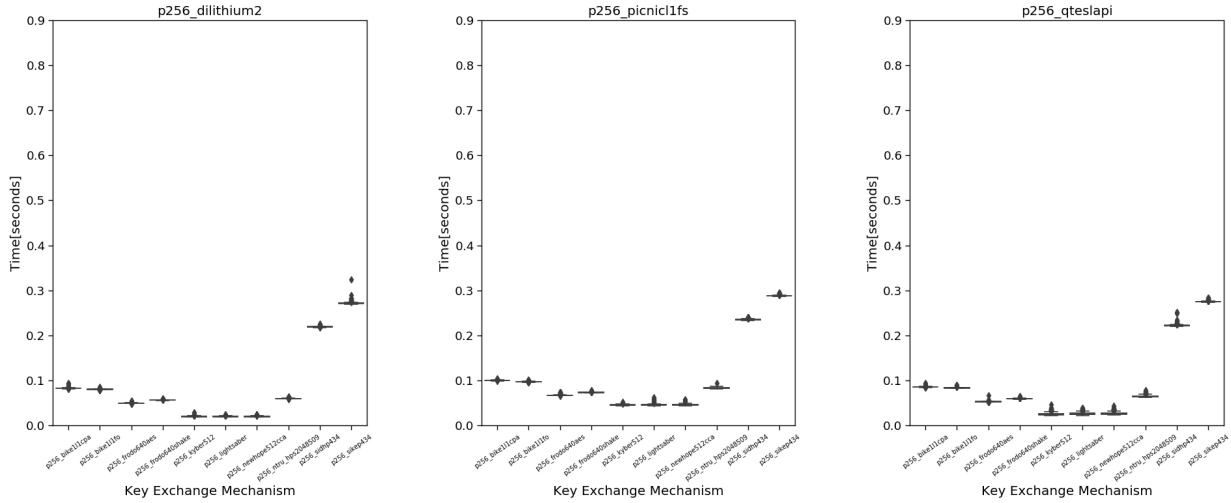
	SAL	Mean	Median	Std. Deviation
p256_dilithium2	I	13.1	12.8	0.7
p256_picnicl1fs	I	28.9	28.9	0.2
p256_qteslapi	I	17.6	16.8	2.4
p384_dilithium4	III	20.3	20.1	0.9
p384_qteslapiii	III	34.2	31.8	6.3
rsa:3072_dilithium2	I	26.5	26.3	0.8
rsa:3072_picnicl1fs	I	42.4	42.3	0.4
rsa:3072_qteslapi	I	31.2	30.2	3.1

Handshake times The time it takes to establish TLS handshakes with Hybrid-PQC can be seen in Table 23 up to 28 for SAL1. Tables 29 and 30 are for SAL3. Note that with SAL3 only the Digital Signature Algorithm is SAL3 and that the Key Exchange Mechanisms are SAL1. SAL1 algorithms with rsa3072 can be seen in Figure 15. SAL1 algorithms with prime256v1 can be seen in Figure 16. The results of SAL3 with prime384v1 can be seen in Figure 17.



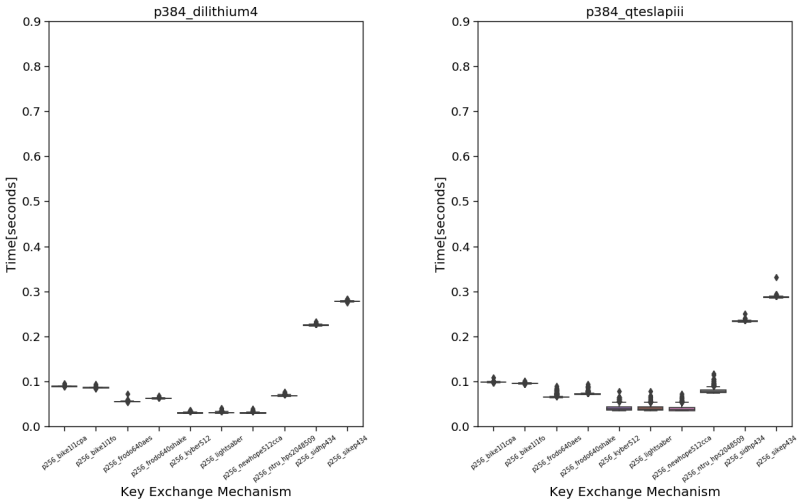
(a) rsa3027_dilithium2 (b) rsa3027_picnicl1fs (c) rsa3072_qteslapi

Figure 15: Measured handshake times for hybrid post quantum cryptography with rsa3072



(a) p256_dilithium2 (b) p256_picnicl1fs.png (c) p256_qteslapi

Figure 16: Measured handshake times for hybrid post quantum cryptography with prime256v1



(a) p384_dilithium4 (b) p384_qteslapiii.png

Figure 17: Measured handshake times for hybrid post quantum cryptography with prime384v1

Level 1 algorithms:

Table 23: rsa3072 with dilithium2 handshake times (ms).

rsa3072_dilithium2 with:	**Mean**	**Median**	**Std. Deviation**
p256_bike1l1cpa	82.6	82.6	0.4
p256_bike1l1fo	80.0	80.0	0.4
p256_frodo640aes	50.3	50.0	1.1
p256_frodo640shake	56.1	55.9	0.8
p256_kyber512	30.0	29.7	1.1
p256_newhope512cca	30.4	29.9	1.8
p256_ntru_hps2048509	68.4	68.0	1.1
p256_lightsaber	30.2	29.9	1.1
p256_sidhp434	218.8	218.5	1.5
p256_sikep434	272.2	271.9	1.3

Table 24: rsa3072 with picnicl1fs handshake times (ms).

rsa3072_picnicl1fs with:	**Mean**	**Median**	**Std. Deviation**
p256_bike1l1cpa	99.4	99.3	0.9
p256_bike1l1fo	97.0	97.0	0.7
p256_frodo640aes	74.9	74.4	1.4
p256_frodo640shake	79.9	80.0	1.3
p256_kyber512	55.7	56.0	1.3
p256_newhope512cca	55.6	56.0	1.3
p256_ntru_hps2048509	93.7	93.9	1.2
p256_lightsaber	55.8	56.2	1.2
p256_sidhp434	235.6	235.4	1.0
p256_sikep434	288.3	288.2	1.0

Table 25: rsa3072 with qteslapi handshake times (ms).

rsa3072_qteslapi with:	**Mean**	**Median**	**Std. Deviation**
p256_bike1l1cpa	85.8	85.7	0.5
p256_bike1l1fo	83.3	83.2	0.5
p256_frodo640aes	56.5	55.6	3.4
p256_frodo640shake	60.8	59.7	2.5
p256_kyber512	35.9	35.2	2.8
p256_newhope512cca	36.0	35.4	2.7
p256_ntru_hps2048509	74.2	73.2	3.2
p256_lightsaber	35.9	35.2	2.9
p256_sidhp434	221.8	221.6	1.0
p256_sikep434	274.9	274.6	1.5

Table 26: prime256v1 with dilithium2 handshake times (ms).

prime256v1_dilithium2 with:	**Mean**	**Median**	**Std. Deviation**
p256_bike1l1cpa	82.6	82.6	0.8
p256_bike1l1fo	80.0	79.9	0.6
p256_frodo640aes	49.5	49.5	0.4
p256_frodo640shake	56.3	56.2	0.4
p256_kyber512	19.9	19.7	0.9
p256_newhope512cca	19.8	19.6	0.9
p256_ntru_hps2048509	59.4	59.3	0.5
p256_lightsaber	19.8	19.6	0.8
p256_sidhp434	219.0	218.8	0.8
p256_sikep434	272.0	271.6	2.8

Table 27: rsa3072 with prime256v1 handshake times (ms).

prime256v1_picnicl1fs with:	**Mean**	**Median**	**Std. Deviation**
p256_bike1l1cpa	99.6	99.5	0.6
p256_bike1l1fo	97.0	97.0	0.6
p256_frodo640aes	67.0	66.8	0.9
p256_frodo640shake	73.1	73.0	0.5
p256_kyber512	44.8	44.5	1.0
p256_newhope512cca	45.5	45.9	1.4
p256_ntru_hps2048509	82.8	82.3	1.2
p256_lightsaber	45.9	46.1	1.7
p256_sidhp434	235.1	234.9	0.9
p256_sikep434	288.2	288.0	0.9

Table 28: prime256v1 with qteslapi handshake times (ms).

prime256v1_qteslapi with:	**Mean**	**Median**	**Std. Deviation**
p256_bike1l1cpa	85.7	85.7	0.6
p256_bike1l1fo	83.2	83.2	0.6
p256_frodo640aes	52.7	52.6	0.8
p256_frodo640shake	59.5	59.4	0.6
p256_kyber512	25.6	24.8	2.8
p256_newhope512cca	26.1	25.3	3.0
p256_ntru_hps2048509	64.5	63.5	2.2
p256_lightsaber	25.8	25.1	2.7
p256_sidhp434	222.0	221.6	2.2
p256_sikep434	274.9	274.7	1.1

Level 3 algorithms:
Note that only the Digital Signature Algorithms are SAL3 and that the Key Exchange Mechanisms are SAL1.

Table 29: prime384v1 with dilithium4 handshake times (ms).

prime384v1_dilithium4 with:	**Mean**	**Median**	**Std. Deviation**
p256_bike1l1cpa	88.7	88.7	0.5
p256_bike1l1fo	85.9	85.9	0.6
p256_frodo640aes	55.5	55.4	0.9
p256_frodo640shake	62.2	62.1	0.4
p256_kyber512	30.3	29.9	1.1
p256_newhope512cca	30.1	29.8	1.1
p256_ntru_hps2048509	68.1	67.9	1.1
p256_lightsaber	30.6	30.2	1.2
p256_sidhp434	224.6	224.5	0.9
p256_sikep434	277.7	277.6	0.9

Table 30: prime384v1 with qteslapiii handshake times (ms).

prime384v1_qteslapiii with:	**Mean**	**Median**	**Std. Deviation**
p256_bike1l1cpa	98.2	98.1	0.7
p256_bike1l1fo	95.7	95.6	0.6
p256_frodo640aes	66.3	65.0	3.8
p256_frodo640shake	72.5	71.7	2.9
p256_kyber512	40.9	38.8	6.7
p256_newhope512cca	39.7	37.5	6.1
p256_ntru_hps2048509	79.3	76.6	6.4
p256_lightsaber	40.1	38.1	6.0
p256_sidhp434	234.1	233.9	1.1
p256_sikep434	287.4	287.1	2.2

A.3. Server specifications

The table below gives an overview of the specifications of our test environment.

Table 31: Server specifications of our test environment

	TLS Server	TLS Client
CPU	Intel Xeon L3426 4 Cores @ 1.87GHz 64bit	Intel Xeon L3426 4 Cores @ 1.87GHz 64bit
RAM	4GB	4GB
OS	Ubuntu 18.04 LTS x86_64	Ubuntu 18.04 LTS x86_64
Kernel	4.15.0-74-generic	4.15.0-74-generic
RTT	avg: 0.589 ms mdev: 0.083 ms	