Post-Quantum TLS experiments

Instantiations, transmission requirements, and performance measurements for NIST security levels I, III and V

Thom Wiggers

thom.wiggers@pqshield.com

June 20, 2023

PQShield

This document describes post-quantum TLS experiments that I have done in the context of writing my PhD thesis. It contains significants parts of chapter 11 of my thesis manuscript, which are in turn based on prior work with Douglas Stebila and Peter Schwabe [36, 37].

Note that we are only comparing the sizes and performance characteristics of the schemes in this and other chapters. It can be argued if this is fair: for example, hash-based schemes, which are generally slow and large, are based on assumptions that are considered to be much more conservative than those assumptions on which much-faster lattice-based schemes are based. However, although one might say a particular scheme performs "best", these comparisons are still useful to estimate the cost of more conservative approaches.

The experimental data and the implementation are found at github.com/thomwiggers/kemtls-experiment/ in the branch `thesis`.

Limitations

For the web, it is important to note that there are several additional signatures that need to be considered: I am not including OCSP staples, and SCTs in these TLS handshakes. Additionally, the measurements reported are done over a simulated network environment: results by Bas Westerbaan [41] show that likely there will be more subtle increases in handshake latency than the sudden jumps observed in my tables when an experiment crosses the congestion window threshold.

Suggested reading approach

This document is quite long and has a lot of comparisons. To get the most out of it, I recommend focusing on those scenarios that you find interesting. In section 2, we show the instantiations, that is the combinations of key encapsulation mechanisms (KEMs) and signature schemes, that I have investigated. In the following, I report figures for both unilateraly and mutually-authenticated TLS handshakes, at National Institute of Standards and Technology (NIST) security levels I, III and V. So, if you're interested in what the comparison for NIST security level III would look like for unilaterally authenticated handshakes, skip ahead to section 4.1.

Future work

The NIST signature scheme on-ramp is days from formally kicking off, and none of the signature schemes that have been put forward have yet made it into my comparisons. I am looking forward to the full list of candidates, and I will be investigating their applicability to TLS. Do note that, based on prior experience with the reference implementations of submissions in the post-quantum competition [23], I expect that it will be a while before I will be able to report comprehensive benchmark results.

Contents

1 Measuring post-quantum TLS on an emulated network

This text appears in chapter 10 in my PhD thesis manuscript.

In our experiments we use the example TLS client and server implementations provided by Rustls. We modified the client to connect specified number of times instead of just once. We have also modified the client and server implementations to, e.g., allow specifying cached certificates. We instrumented the handshake implementations to print the number of nanoseconds that have elapsed, starting from either sending or receiving the initial message until operations of interest for both the client and the server.

Part of the measurement setup is a script[1] that prepares all the experiments we are interested in. As the version of Rustls that our implementations are based on hard-codes the client's default ephemeral key-exchange algorithm, we replace the default based on our settings.[2] We then simply compile the example TLS server and client applications for every different key exchange method. We also generate the certificates necessary for the experiment. To make sure all of this is reproducible, we execute all these steps in an isolated Docker container.[3] This fixes the Rust compiler version and isolates the compilation from the host operating system. As Rust statically links binaries, we can use the binaries generated in the container without having to be too careful about keeping in sync with a dynamically linked TLS library.

We follow the same methodology as [30] for setting up emulated networks.[4] The measurements are done using the Linux kernel's network namespacing [9] and network emulation (`NetEm`) features [17]. We create network namespaces for the clients and the servers and create virtual network interfaces in

[1] Please refer to measuring/scripts/experiment.py in our experiment codebase.
[2] Refer to measuring/scripts/create-experimental-setup.sh.
[3] Refer to Dockerfile in the repository root.
[4] Refer to measuring/scripts/setup_ns.sh.

those namespaces. We vary the latency and bandwidth of the emulated network. `NetEm` adds a latency to the *outgoing* packets, so to add a latency of x ms, we add $x/2$ ms of latency to the client and server interfaces; following [30], we consider round-trip times (RTTs) of 30.9 ms (representing an transcontinental connection) and 195.5 ms (representing a transpacific connection). We also throttle the bandwidth of the virtual interfaces, considering both 1000 Mbps and 10 Mbps connections. We do not vary the packet loss rate, fixing it at 0 %.

We ran measurements on a server with two Intel Xeon Gold 6230 (Cascade Lake) CPUs, each featuring 20 physical cores, which gives us 80 hyperthreaded cores in total. For the measurements, we run 40 clients and servers in parallel, such that each process has its own (hyperthreaded) core. We measured 20 000 handshakes for each scheme and set of network parameters.

We also report the sizes of the experimental handshakes. To obtain these numbers, we run the generated TLS client and server with the certificates relevant to the handshake over localhost.[5] We record and process the transmitted TCP packets using `tshark` [42], and use a small Python script to extract the handshake metrics.

1.1 Measured network scenarios

In our experiments on the emulated network, we simulate two network environments, following the choices made in [30]. The first environment, which represents a high-bandwidth transcontinental connection, uses a network round-trip latency of 30.9 ms and a network bandwidth of 1000 Mbps. The second environment resembles a transpacific, low-bandwidth connection and uses a network round-trip latency of 195.5 ms and has a bandwidth of 10 Mbps. We do not vary the packet loss rates, as this would mostly affect the results at higher percentiles which we do not report.

2 Selecting algorithms for experiments

There are many post-quantum KEM and signature schemes that we could use for our experiments. We select some instantiations that we think are interesting, which we will introduce in this section. As a baseline, we use an instantiation based on elliptic-curve Diffie–Hellman (ECDH) and RSA. We mainly use the algorithms selected for standardization in the NIST post-quantum cryptography (PQC) standardization project, as well as the remaining round-4 finalists for KEMs [1]. Separately from the NIST PQC standardization project, NIST and the Internet Engineering Task Force (IETF) have already standardized stateful hash-based signature schemes XMSS [14, 18] and LMS [14, 27]. These stateful hash-based signature schemes are as conservative as SPHINCS$^+$ but much smaller, so we will present some instantiations that make use of XMSS^{MT}. However, as their stateful nature makes them very sensitive to user error, we restrict their use to certificate authority (CA) certificates. As XMSS only has standardized parameter sets at above NIST PQC security level V, we selected customized parameters for use in CA certificates at the different security levels in section 7.

2.1 Instantiations of post-quantum TLS 1.3

In each instantiation, we select:

1. an algorithm for ephemeral key exchange, negotiated by the TLS 1.3 client and server;
2. an algorithm for handshake authentication, used in the server's certificate;

[5]Refer to `measuring/scripts/measure-handshake-size.sh`.

3. an algorithm for authentication of the server's certificate by the (intermediate) CA certificate, which we may assume the client to already have;

4. an algorithm to authenticate the intermediate CA certificate by a root CA certificate, which is always assumed to be preinstalled.

For TLS handshakes that use mutual authentication, we additionally select:

5. an algorithm for client authentication, used in the client certificate;

6. an algorithm for authentication of the client certificate by a CA certificate, which is assumed to be preinstalled.

In our experiments, we will try to showcase how algorithms in the NIST PQC standardization project perform, as well as highlight how some careful choices for certificate algorithms can make large differences. We will use the following scenarios:

Pre-quantum The pre-quantum instantiation uses X25519 [7] for key exchange and RSA-2048 [32] for all signatures.

Primary For ephemeral key exchange, this instantiation uses Kyber [35], the only KEM which was selected for standardization for post-quantum key exchange. Dilithium [26], the algorithm which, when it was selected for standardization, was named the *primary* algorithm for post-quantum signatures, is used for all signatures.

Falcon This instantiation uses Kyber for ephemeral key exchange, and Falcon [31] for all signatures. Falcon was also selected for standardization by NIST but its use is not recommended unless its implementation concerns can be properly addressed: Falcon is very sensitive to side channels and requires constant-time 64-bit floating-point operations for signing.

Falcon offline This instantiation uses Kyber for ephemeral key exchange and Dilithium for the (online) handshake signatures of the server and, if mutually authenticating, the client. The CA signatures in certificates are instantiated using Falcon. Because these can be produced offline by the CA, we can assume they can mitigate all implementation concerns: signature verification does not have Falcon's implementation considerations.

SPHINCS$^+$-f This instantiation uses Kyber for ephemeral key exchange. For all signatures SPHINCS$^+$ [20] is used, which is the only NIST selection for standardization that is not based on lattice assumptions. Specifically, this instantiation uses the *fast* variant of SPHINCS$^+$, which has faster runtime but larger signatures. For the hash function, we use Haraka, as explained in section 10.6.

SPHINCS$^+$-s This instantiation is like the SPHINCS$^+$-f-variant, but it uses the *small* variant of SPHINCS$^+$. This variant requires more computation time for signing but has significantly smaller signatures.

Hash-based signatures This conservative instantiation uses Kyber for ephemeral key exchange and SPHINCS$^+$ for the handshake authentication signatures. To minimize the size, we use a custom instantiation of XMSS^{MT} at the appropriate NIST security level for the CA signatures. These instances, which we label XMSS$_s^{MT}$, are described in section 7.

Hash-based CA This instantiation uses Kyber for key exchange and Dilithium for the handshake authentication signatures. To minimize the size, we use a custom instantiation of XMSS^{MT} at the appropriate NIST security level for the CA signatures.

HQC This instantiation uses HQC, a round-4 KEM candidate in the NIST PQC standardization project, for ephemeral key exchange. HQC relies on assumptions based on decoding of quasi-cyclic codes, instead of on assumptions on lattices. For handshake authentication and CA signatures, Dilithium is used.

Note that for presentation purposes, we will refer to these scenarios in our tables and figures by handles, which are composed of the first letters of each of the selected algorithms (though X25519 is represented by the letter 'e' for ECDH). As an example, we denote by KDDD the instantiation that uses Kyber for ephemeral key exchange and uses Dilithium for server authentication and the intermediate and root CA certificates. For an overview of these handles, refer to the tables that show the communication sizes, e.g. table 1.

The remaining candidates for post-quantum key exchange in round 4 of the NIST PQC standardization project, are unfortunately not suitable for our experiments. BIKE [5] does not have IND-CCA-secure parameters available, and the public keys of Classic McEliece [2] are too large to use in TLS 1.3.

Note that the use of intermediate CA certificates is not required. Alternatively, there exist proposals in which the intermediate certificate can be cached: the client can then request intermediate CA certificates to not be transferred [22]. To represent these scenarios, we will also show results for experiments that use the intermediate certificate as the root certificate, and thus do not transmit or verify the root certificate.

3 Instantiation and results at NIST level I

We measured and compare the performance of TLS 1.3 at NIST security level I. This security level offers security comparable to that given by AES-128. As it is the lowest security level, the parameter sets are the most aggressively chosen. They generally offer the smallest public key, ciphertext, and signature sizes and the shortest computation times. First, we will cover unilaterally authenticated handshakes, in which only the server is authenticated by a certificate and a signature. This scenario is very important to the web, as this is the handshake mode that is almost exclusively used by web browsers [10]. Afterward, we will discuss mutually authenticated handshakes, in which the client also presents a certificate of their identity and signs the handshake. Although this setting is not very relevant to web browsing, it is for example used to secure service-to-service communication or in VPNs.

3.1 Unilaterally authenticated TLS 1.3

Communication requirements

In table 1, we show the communication sizes of our choices of instantiations. We also give the abbreviated handles by which we will refer to the instantiations in other tables. We give the sum of the data necessary for the ephemeral key exchange, the handshake authentication signature, and the leaf certificate, as this is the minimum amount of data, excluding protocol overhead, that needs to be transferred if no intermediate CA certificates are required. In these experiments, the intermediate CA certificate is assumed to be used as the trusted root certificate. We also give the total amount of public key data that is transmitted when an intermediate CA certificate is included.

Post-quantum ephemeral key exchange requires much more data than ECDH, and post-quantum signatures have much larger public key and signature sizes than RSA-2048. Falcon is the smallest general-purpose post-quantum signature scheme, while Dilithium is much larger. Finally, we see that the schemes that do not rely on lattice assumptions are much larger than their lattice equivalents. Using HQC-128 instead of Kyber-512 for key exchange requires 5162 additional bytes. The variants based on hash-based signatures also require much more data. The only exception is our custom XMSS parameter set, which appears as an attractive option to reduce the size of the certificate chain when used for CA certificates: using $\text{XMSS}_s^{\text{MT}}$-I

Table 1: Instantiations at NIST level I of unilaterally authenticated post-quantum TLS handshakes and the sizes of the public-key cryptography elements in bytes.

		Leaf certificate			**Int. CA certificate**			**Offline**
Experiment handle	**Key Exchange** pk+ct	**Handshake auth.** pk+sig	**Int. CA signature** sig	**Sum**	**Int. CA public key** pk	**Root CA signature** sig	**Sum**	**Root CA public key** pk
Pre-quantum errr	X25519 64	RSA-2048 528	RSA-2048 256	848	RSA-2048 272	RSA-2048 256	1 376	RSA-2048 272
Primary KDDD	Kyber-512 1568	Dilithium2 3732	Dilithium2 2420	7 720	Dilithium2 1312	Dilithium2 2420	11 452	Dilithium2 1312
Falcon KFFF	Kyber-512 1568	Falcon-512 1563	Falcon-512 666	3 797	Falcon-512 897	Falcon-512 666	5 360	Falcon-512 897
Falcon offline KDFF	Kyber-512 1568	Dilithium2 3732	Falcon-512 666	5 966	Falcon-512 897	Falcon-512 666	7 529	Falcon-512 897
SPHINCS$^+$-f KSfSfSf	Kyber-512 1568	SPHINCS$^+$-128f 17 120	SPHINCS$^+$-128f 17 088	35 776	SPHINCS$^+$-128f 32	SPHINCS$^+$-128f 17 088	52 896	SPHINCS$^+$-128f 32
SPHINCS$^+$-s KSsSsSs	Kyber-512 1568	SPHINCS$^+$-128s 7888	SPHINCS$^+$-128s 7856	17 312	SPHINCS$^+$-128s 32	SPHINCS$^+$-128s 7856	25 200	SPHINCS$^+$-128s 32
Hash-based signatures KSsXX	Kyber-512 1568	SPHINCS$^+$-128s 7888	$\text{XMSS}_s^{\text{MT}}$-I 979	10 435	$\text{XMSS}_s^{\text{MT}}$-I 32	$\text{XMSS}_s^{\text{MT}}$-I 979	11 446	$\text{XMSS}_s^{\text{MT}}$-I 32
HBS-CA KDXX	Kyber-512 1568	Dilithium2 3732	$\text{XMSS}_s^{\text{MT}}$-I 979	6 279	$\text{XMSS}_s^{\text{MT}}$-I 32	$\text{XMSS}_s^{\text{MT}}$-I 979	7 290	$\text{XMSS}_s^{\text{MT}}$-I 32
HQC HDDD	HQC-128 6730	Dilithium2 3732	Dilithium2 2420	12 882	Dilithium2 1312	Dilithium2 2420	16 614	Dilithium2 1312

in place of SPHINCS$^+$-128s saves 6877 bytes of data (87.5 %). Using XMSS$_s^{MT}$-I in place of Dilithium2 saves 1441 bytes of data (59.5 %).

Computational requirements

In table 2, we compare the amount of computation that each combination of algorithms requires. The amount given for client operations is the sum of the key generation and decapsulation operations for the ephemeral key exchange, the verification time of the handshake signature, the verification time for the leaf certificate, and if an intermediate certificate is transmitted, the verification time of the intermediate certificate. For reference, the time of individual key-generation, signing, verification, encapsulation, and decapsulation operations are given in tables 10.1 and 10.2.

Table 2: Computation time in ms for asymmetric cryptography at NIST level I for each of the unilaterally authenticated post-quantum TLS instantiations at the client and server.

Handle	Intermediate cert. as root			With intermediate CA cert.		
	Client	Server	Sum	Client	Server	Sum
errr	0.134	0.629	0.763	0.150	0.629	0.779
KDDD	0.166	0.204	0.370	0.230	0.204	0.434
KFFF	0.320	0.668	0.988	0.461	0.668	1.129
KDFF	0.243	0.204	0.447	0.384	0.204	0.588
KSfSfSf	0.924	5.544	6.468	1.367	5.544	6.911
KSsSsSs	0.218	60.542	60.760	0.308	60.542	60.850
KSsXX	8.360	60.542	68.902	16.592	60.542	77.134
KDXX	8.334	0.204	8.538	16.566	0.204	16.770
HDDD	0.461	0.317	0.778	0.525	0.317	0.842

Comparing the lattice-based experiments KDDD and KFFF with the pre-quantum errr instantiation, it is evident that while post-quantum cryptography may be bigger, it is not necessarily also *slower*: KFFF performs comparable with errr, while KDDD uses much less computation time. However, this does not hold for all schemes. While HQC requires only slightly more computation time than the Kyber-based experiment, the experiments that use hash-based signatures require much more time. But also between the hash-based schemes there exist large differences. The smaller variant of SPHINCS$^+$ requires vastly more computation: to produce the handshake signature with SPHINCS$^+$-128s instead of SPHINCS$^+$-128f requires 54.998 ms more computation (90.9 %). Our custom XMSS$_s^{MT}$-I parameters have been tuned for as small a signature as possible, and this is also clearly visible in the computation time: the cost of verifying XMSS signatures adds significantly to the client's computation time.

Handshake performance

In table 3, we can see the average times taken in the handshakes instantiated with our selected algorithms for a high-bandwidth, low-latency connection, using a latency of 30.9 ms and a 1000 Mbps link speed. We again give times for when the intermediate CA certificate algorithm is not transmitted and thus used as a root CA certificate, and for the scenario in which the intermediate CA does need to be transmitted and verified. In our experiments, we assume an HTTP-like scenario in which the client requests some data from the server, so the server needs to receive the client's request before it can start transmitting application traffic. In each of these scenarios, we give the amount of time until the client is ready to send a request (i.e., in TLS 1.3, the client has received `ServerFinished` and sent `ClientFinished`), the time until the client receives the response to its request from the server, and the time until the server has completed the handshake (i.e., in TLS 1.3, received `ClientFinished`). Note that the timers of the client and the server are independent,

Table 3: Average handshake times in ms for unilaterally authenticated post-quantum TLS experiments at NIST level I with 30.9 ms latency and 1000 Mbps bandwidth. Server and client timers are independent.

Handle	Intermediate cert. as root			With root certificate		
	Client sent req.	Client recv. resp.	Server HS done	Client sent req.	Client recv. resp.	Server HS done
errr	65.9	97.0	35.0	66.1	97.2	35.2
KDDD	63.6	94.8	32.7	63.9	95.0	33.0
KFFF	64.6	95.8	33.7	65.0	96.1	34.0
KDFF	63.7	94.8	32.8	64.0	95.2	33.1
KSfSfSf	106.4	137.6	75.5	136.9	168.1	106.0
KSsSsSs	166.7	197.7	135.8	166.9	198.0	136.0
KSsXX	155.5	186.6	124.6	171.1	202.1	140.1
KDXX	97.2	128.3	66.2	113.7	144.8	82.8
HDDD	63.6	94.7	32.7	63.9	95.1	33.0

Table 4: Average handshake times in ms for unilaterally authenticated post-quantum TLS experiments at NIST level I with 195.5 ms latency and 10 Mbps bandwidth. Server and client timers are independent.

Handle	Intermediate cert. as root			With root certificate		
	Client sent req.	Client recv. resp.	Server HS done	Client sent req.	Client recv. resp.	Server HS done
errr	397.1	593.1	201.3	397.7	593.7	201.8
KDDD	405.5	602.8	208.8	410.3	610.0	212.8
KFFF	397.3	593.3	200.8	399.5	595.5	203.0
KDFF	398.7	594.7	202.2	405.6	602.8	208.8
KSfSfSf	1177.3	1544.4	963.5	1751.4	2038.1	1524.6
KSsSsSs	914.0	1116.4	715.3	979.2	1217.9	772.5
KSsXX	530.6	735.6	331.2	564.9	776.6	364.9
KDXX	446.9	648.1	240.6	475.4	679.7	266.1
HDDD	405.5	602.7	208.7	410.5	610.1	212.9

and start counting as soon as the `ClientHello` message is constructed (for the client) or received (for the server).

For the instantiations that use pre-quantum cryptography, or any combination of the fast algorithms Kyber-512, HQC-128, Dilithium2, and Falcon-512, we see that the handshake times are roughly a multiple of the connection latency. The client can transmit its request after two times the handshake latency, which matches the two round-trips necessary: one for the TCP connection establishment, and the single round-trip in which the TLS handshake is completed. The response is received in the return round-trip after sending off the request. The server receives `ClientFinished` and thus completes its part of the handshake a single round-trip after transmitting the `ServerFinished` packet. The computational requirements of these algorithms on the chosen platforms are so small that they do not meaningfully contribute to the connection establishment times. With the connection parameters in this experiment, the additional amounts of traffic required for Dilithium2 compared to Falcon-512 or HQC-128 compared to Kyber-512 do not meaningfully contribute to the handshake time.

In the KDXX parameter set which uses XMSS$_s^{MT}$-I for the CA certificates to reduce the amount of handshake traffic, the additional computation time required to verify the XMSS signatures adds significantly to the handshake time. The additional latency for the experiment in which the root CA is omitted, compared to the KDDD experiment, is more than the 8.2 ms that are required on average to verify the XMSS$_s^{MT}$-I signature on the server's leaf certificate. We suspect this happens because the additional verification time interacts with the TCP congestion control algorithms. We see a similar delay in the KSsSsSs parameter set, but this instantiation additionally suffers from the larger amount of bytes that need to be transmitted. The amount of data for this selection of algorithms exceeds the initial congestion window (`initcwnd`) set in the TCP slow start algorithm [11], which is the initial limit on the amount of data (measured in maximum segment size (MSS)) that can be sent on a TCP connection before receiving an acknowledgment packet. The default `initcwnd` on Linux is 10 MSS, which means that after transmitting about 14.5 kB, the server needs to wait for the client to acknowledge the packets that it has received before it will send more. This induces extra round-trip delays. As the instance using SPHINCS$^+$-128f has very large certificates due to the large signature size of SPHINCS$^+$-128f, these extra round-trips slow down the connection establishment despite the much faster signing time compared to SPHINCS$^+$-128s. Still, for this high-bandwidth, low-latency connection, the SPHINCS$^+$-128f instance is much faster than any instance using SPHINCS$^+$-128s: the computational requirements are just too large compared to the communication overhead. When including the intermediate CA certificate, KSfSfSf requires 29.9 ms (15.1 %) less time than KSsSsSs before the client receives the server's response.

In table 4, we compare the same metrics for experiments on a high-latency, low-bandwidth connection, using a latency of 195.5 ms latency and 10 Mbps connection bandwidth. With these connection characteristics, we see that the sizes of the public keys, ciphertexts, and signatures start to matter more. KFFF, which has the smallest sum of public-key cryptography objects, has the best performance, coming very close to the performance of the pre-quantum errr instantiation. Comparatively, KDDD, which suffers from the much larger Dilithium2 public keys and signatures, has higher connection establishment times. When including the intermediate CA certificate, KDDD takes an additional 16.3 ms (2.7 %) before the client receives the response from the server, compared to the pre-quantum instance. The performance of the Kyber-512/SPHINCS$^+$-128f instantiation KSfSfSf again clearly suffers under the weight of SPHINCS$^+$-128f signatures, now showing the worst performance of all parameter sets.

The two instantiations that use Kyber-512 and Dilithium2 for the online components of the TLS 1.3 handshake, but rely on different algorithms for CA certificates, KDFF and KDXX, appear promising: by reducing the amount of handshake traffic, they perform fairly reasonably well (though the performance of XMSS$_s^{MT}$-I still hurts KDXX). Especially KDFF, a combination of two to-be-standardized algorithms, seems to combine the best of both worlds: using Dilithium2 for the online handshake authentication avoids the implementation concerns associated with Falcon-512 signature generation while offering performance on par with the all-Falcon KFFF instantiation but even KDXX is only slightly slower.

3.2 Mutually authenticated TLS 1.3

Communication requirements

Table 5 shows the precise selection of algorithms for our instantiations of mutually authenticated post-quantum TLS 1.3. Like for the unilaterally authenticated instantiations, we also show the sizes of the public-key cryptographic elements necessary for ephemeral key exchange, server authentication, and client authentication, and the total. (For a better presentation, we only show the sum of the size of the public key and the signature that are part of each certificate, even if they use different algorithms). Again, we examine two scenarios for each instantiation, one that omits an intermediate CA certificate (and thus uses it as the trusted root certificate) and one that makes use of intermediate CA certificates, transmitting them during the handshake.

The difference in size between the unilaterally authenticated handshakes in table 1 and the mutually authenticated handshakes is exactly the number of bytes listed in the client authentication column. As the size of the Kyber-512 key exchange is much smaller than the sum of a public key and signature used in most of our instantiations, we roughly double the sizes of the handshakes when omitting intermediate certificates.

Computational requirements

Table 6 shows the amount of computation required for the cryptographic operations using the algorithms in our instantiations. As the client and the server now both need to produce a signature during the handshake and verify a certificate chain, they need to perform the same amount of work when an intermediate CA certificate is not used. Otherwise, the client needs to additionally verify this certificate. As signing is much more expensive than verifying for most algorithms, we see that the total computational load roughly doubles compared to the unilaterally authenticated experiments.

Handshake performance

In tables 7 and 8, we show the performance of our instantiations of the mutually authenticated TLS 1.3 running on a 30.9 ms latency, 1000 Mbps network and on a 195.5 ms latency, 10 Mbps network. Note that even though the transmission size of some of the instances, when including client authentication, exceeds 15 kB, we point out that the initial congestion window is not shared between the client and the server: thus the client is free to send as much data as it can fit in its congestion window regardless of the size of the server's certificate. Furthermore, the client needs to fully receive the server's certificate and verify the server's handshake signature before it may send the client certificate. This means that TCP congestion control has a chance to catch up and the client's certificate size does not contribute as much to the connection congestion; unlike the server's certificate which is all sent out immediately as the connection is established. Additionally, as the client sends out its request immediately after it transmits its certificate, the server first needs to process the client certificate before it can process the client request. This is visible in our measurements as an increased gap between the client sending its request and receiving the response. We see that large certificates, such as in the instance based on SPHINCS$^+$-128f, especially contribute to longer waiting times before the client receives its response from the server. The KSfSfSf-SfSf experiment is 44.8 ms (32.6 %) slower than the unilaterally authenticated KSfSfSf experiment when omitting intermediate CA certificates on the low-latency network. On the slow network, comparing the same two SPHINCS$^+$-128f experiments shows a 452.7 ms (29.3 %) difference. Note that the relative difference is very similar (while naively we would expect the larger size to take longer), which is due to the TCP congestion control algorithm already having had a chance to scale the client's bandwidth before the client certificate is transmitted: the large server certificate results in acknowledgments being sent out that influence congestion control.

Table 5: Instantiations at NIST level I of mutually authenticated post-quantum TLS experiments and the sizes of the public-key cryptography elements transmitted in bytes.

Experiment	**Key**	**Server**	**Client**	**Sum**	**Int. CA**	**Sum**
handle	**exchange**	**authentication**	**authentication**		**certificate**	
Pre-quantum	X25519	hs:RSA-2048	hs:RSA-2048		pk:RSA-2048	
		sig:RSA-2048	sig:RSA-2048	1 632	sig:RSA-2048	2 160
errr-rr	64	784	784		528	
Primary	Kyber-512	hs:Dilithium2	hs:Dilithium2		pk:Dilithium2	
		sig:Dilithium2	sig:Dilithium2	13 872	sig:Dilithium2	17 604
KDDD-DD	1568	6152	6152		3732	
Falcon	Kyber-512	hs:Falcon-512	hs:Falcon-512		pk:Falcon-512	
		sig:Falcon-512	sig:Falcon-512	6 026	sig:Falcon-512	7 589
KFFF-FF	1568	2229	2229		1563	
Falcon offline	Kyber-512	hs:Dilithium2	hs:Dilithium2		pk:Falcon-512	
		sig:Falcon-512	sig:Falcon-512	10 364	sig:Falcon-512	11 927
KDFF-DF	1568	4398	4398		1563	
SPHINCS$^+$-f	Kyber-512	hs:SPHINCS$^+$-128f	hs:SPHINCS$^+$-128f		pk:SPHINCS$^+$-128f	
		sig:SPHINCS$^+$-128f	sig:SPHINCS$^+$-128f	69 984	sig:SPHINCS$^+$-128f	87 104
KSfSfSf-SfSf	1568	34 208	34 208		17 120	
SPHINCS$^+$-s	Kyber-512	hs:SPHINCS$^+$-128s	hs:SPHINCS$^+$-128s		pk:SPHINCS$^+$-128s	
		sig:SPHINCS$^+$-128s	sig:SPHINCS$^+$-128s	33 056	sig:SPHINCS$^+$-128s	40 944
KSsSsSs-SsSs	1568	15 744	15 744		7888	
Hash-based	Kyber-512	hs:SPHINCS$^+$-128s	hs:SPHINCS$^+$-128s		pk:XMSS_s^{MT}-I	
signatures		sig:XMSS_s^{MT}-I	sig:XMSS_s^{MT}-I	19 302	sig:XMSS_s^{MT}-I	20 313
KSsXX-SsX	1568	8867	8867		1011	
HBS-CA	Kyber-512	hs:Dilithium2	hs:Dilithium2		pk:XMSS_s^{MT}-I	
		sig:XMSS_s^{MT}-I	sig:XMSS_s^{MT}-I	10 990	sig:XMSS_s^{MT}-I	12 001
KDXX-DX	1568	4711	4711		1011	
HQC	HQC-128	hs:Dilithium2	hs:Dilithium2		pk:Dilithium2	
		sig:Dilithium2	sig:Dilithium2	19 034	sig:Dilithium2	22 766
HDDD-DD	6730	6152	6152		3732	

hs: certificate public key and handshake signature
pk: certificate public key   sig: certificate signature

Table 6: Computation time in ms for asymmetric cryptography at NIST level I for each of the mutually authenticated post-quantum TLS instantiations at the client and server.

Handle	**Intermediate cert. as root**			**With intermediate CA cert.**		
	Client	Server	Sum	Client	Server	Sum
errr-rr	0.660	0.661	1.321	0.676	0.661	1.337
KDDD-DD	0.344	0.332	0.676	0.408	0.332	0.740
KFFF-FF	0.962	0.950	1.912	1.103	0.950	2.053
KDFF-DF	0.421	0.409	0.830	0.562	0.409	0.971
KSfSfSf-SfSf	6.442	6.430	12.872	6.885	6.430	13.315
KSsSsSs-SsSs	60.734	60.722	121.456	60.824	60.722	121.546
KSsXX-SsX	68.876	68.864	137.740	77.108	68.864	145.972
KDXX-DX	8.512	8.500	17.012	16.744	8.500	25.244
HDDD-DD	0.639	0.445	1.084	0.703	0.445	1.148

Table 7: Average handshake times in ms for mutually authenticated post-quantum TLS experiments at NIST level I with 30.9 ms latency and 1000 Mbps bandwidth. Server and client timers are independent.

	Intermediate cert. as root			**With root certificate**		
Handle	**Client sent req.**	**Client recv. resp.**	**Server HS done**	**Client sent req.**	**Client recv. resp.**	**Server HS done**
errr-rr	68.9	100.3	38.3	68.9	100.4	38.3
KDDD-DD	64.2	95.8	33.7	64.4	96.0	34.0
KFFF-FF	66.1	97.9	35.8	66.4	98.2	36.1
KDFF-DF	64.2	96.0	33.9	64.6	96.3	34.2
KSfSfSf-SfSf	117.8	182.3	120.2	148.2	212.8	150.7
KSsSsSs-SsSs	247.9	310.1	248.2	248.2	310.5	248.5
KSsXX-SsX	213.1	254.8	192.9	221.4	263.1	201.2
KDXX-DX	98.7	160.2	98.2	113.4	170.5	108.5
HDDD-DD	64.2	95.8	33.8	64.5	96.1	34.1

Table 8: Average handshake times in ms for mutually authenticated post-quantum TLS experiments at NIST level I with 195.5 ms latency and 10 Mbps bandwidth. Server and client timers are independent.

	Intermediate cert. as root			**With root certificate**		
Handle	**Client sent req.**	**Client recv. resp.**	**Server HS done**	**Client sent req.**	**Client recv. resp.**	**Server HS done**
errr-rr	399.9	597.2	205.2	401.0	598.4	206.1
KDDD-DD	404.5	606.9	212.9	412.0	617.7	217.9
KFFF-FF	399.1	597.8	205.3	402.8	601.6	208.8
KDFF-DF	399.8	599.9	207.3	405.1	606.2	211.7
KSfSfSf-SfSf	1224.6	1997.1	1397.7	1598.9	2339.3	1660.3
KSsSsSs-SsSs	849.3	1292.2	877.1	864.8	1296.9	873.4
KSsXX-SsX	583.5	801.8	398.5	594.9	812.3	406.9
KDXX-DX	440.4	671.2	272.1	463.5	692.3	292.1
HDDD-DD	403.2	605.5	211.8	411.3	616.7	217.2

4 Instantiation and results at NIST level III

In this section, we measure and compare the performance of TLS 1.3 at NIST PQC security level III. This security level corresponds to, roughly, AES-192 in terms of security. This category is significant, as, among others, the authors of Kyber and Dilithium have recommended using the level-III instantiations of their schemes [15, 24]. Again, we first examine unilaterally authenticated handshakes, before we look at mutually authenticated handshakes.

4.1 Unilaterally authenticated TLS 1.3

Table 9: Instantiations at NIST level III of unilaterally authenticated post-quantum TLS handshakes and the sizes of the public-key cryptography elements in bytes.

		Leaf certificate			**Int. CA certificate**			**Offline**
Experiment handle	**Key Exchange** pk+ct	**Handshake auth.** pk+sig	**Int. CA signature** sig	**Sum**	**Int. CA public key** pk	**Root CA signature** sig	**Sum**	**Root CA public key** pk
Pre-quantum errr	X25519 64	RSA-2048 528	RSA-2048 256	848	RSA-2048 272	RSA-2048 256	1 376	RSA-2048 272
Primary KDDD	Kyber-768 2272	Dilithium3 5245	Dilithium3 3293	10 810	Dilithium3 1952	Dilithium3 3293	16 055	Dilithium3 1952
Falcon KFFF	Kyber-768 2272	Falcon-1024 3073	Falcon-1024 1280	6 625	Falcon-1024 1793	Falcon-1024 1280	9 698	Falcon-1024 1793
Falcon offline KDFF	Kyber-768 2272	Dilithium3 5245	Falcon-1024 1280	8 797	Falcon-1024 1793	Falcon-1024 1280	11 870	Falcon-1024 1793
SPHINCS$^+$-f KSfSfSf	Kyber-768 2272	SPHINCS$^+$-192f 35 712	SPHINCS$^+$-192f 35 664	73 648	SPHINCS$^+$-192f 48	SPHINCS$^+$-192f 35 664	109 360	SPHINCS$^+$-192f 48
SPHINCS$^+$-s KSsSsSs	Kyber-768 2272	SPHINCS$^+$-192s 16 272	SPHINCS$^+$-192s 16 224	34 768	SPHINCS$^+$-192s 48	SPHINCS$^+$-192s 16 224	51 040	SPHINCS$^+$-192s 48
Hash-based signatures KSsXX	Kyber-768 2272	SPHINCS$^+$-192s 16 272	XMSS_s^{MT}-III 1851	20 395	XMSS_s^{MT}-III 48	XMSS_s^{MT}-III 1851	22 294	XMSS_s^{MT}-III 48
HBS-CA KDXX	Kyber-768 2272	Dilithium3 5245	XMSS_s^{MT}-III 1851	9 368	XMSS_s^{MT}-III 48	XMSS_s^{MT}-III 1851	11 267	XMSS_s^{MT}-III 48
HQC HDDD	HQC-192 13 548	Dilithium3 5245	Dilithium3 3293	22 086	Dilithium3 1952	Dilithium3 3293	27 331	Dilithium3 1952

Communication requirements
In table 9, we show the communication sizes of our choices of instantiations. Comparing the instantiations with those at NIST level I, we see that the sizes become significantly larger across all schemes used in the

instantiations. For example, Kyber-768 requires 704 bytes (44.9 %) more transmission. As Dilithium2, which we used in our level-I experiments, already has security level II, we see a modest increase of 1513 bytes (40.5 %) for its public key and signature combined. For all of the other schemes, we get around 50 % increase in sizes; though it should be noted that Falcon-1024 offers NIST security level V.

Table 10: Computation time in ms for asymmetric cryptography at NIST level III for each of the unilaterally authenticated post-quantum TLS instantiations at the client and server.

Handle	Intermediate cert. as root			With intermediate CA cert.		
	Client	Server	Sum	Client	Server	Sum
errr	0.134	0.629	0.763	0.150	0.629	0.779
KDDD	0.203	0.830	1.033	0.284	0.830	1.114
KFFF	0.385	0.813	1.198	0.557	0.813	1.370
KDFF	0.294	0.830	1.124	0.466	0.830	1.296
KSfSfSf	0.841	6.261	7.102	1.241	6.261	7.502
KSsSsSs	0.311	118.885	119.196	0.446	118.885	119.331
KSsXX	12.075	118.885	130.960	23.974	118.885	142.859
KDXX	12.021	0.830	12.851	23.920	0.830	24.750
HDDD	1.190	1.263	2.453	1.271	1.263	2.534

Computational requirements

When comparing the computation requirements of these instantiations as listed in table 10, we see that although instances based on the lattice-based schemes Kyber, Dilithium, and Falcon see significant increases in computation time (Falcon takes about two times as much time), they still do not require much more time than the original pre-quantum instantiation using X25519 and RSA-2048. The two SPHINCS$^+$-192 variants also take around twice as much time, but as the amount of time taken was already quite large, we now see the server requires over 100 ms just for cryptographic computations in the instances using SPHINCS$^+$-192s for handshake authentication.

Handshake performance

This comes together in the average handshake timings shown in tables 11 and 12. Comparing the computation times, we see that the KFFF instance performs slightly worse than the KDDD instance. Otherwise, we see that the instances largely follow the pattern established in the level-I experiments.

For the high-bandwidth connection, we see that again the sizes largely do not matter for all instantiations that stay under the limits of the TCP slow start algorithm. However, at NIST security level III, more instantiations now do exceed this limit. In the KDDD and HDDD instances in which an intermediate CA certificate is transferred, the penalty of an additional round-trip can be seen in the connection establishment times due to the large size of the certificates exceeding the approximately 14.5 kB congestion window transmission limit. When the intermediate CA certificate is used as root CA certificate, the `ServerCertificate` stays under the limit and the penalty is avoided.

In the experiments using the variants of SPHINCS$^+$-192, we see that the large sizes of the public keys and certificates affect the handshake performance. The SPHINCS$^+$-192s handshake has large increases in the handshake times as both the computation time and amount of data increase dramatically. The SPHINCS$^+$-192f handshake experiment including an intermediate CA certificate on the 30.9 ms latency, 1000 Mbps network takes 41.7 ms (24.8 %) longer before the client received the response at level III than at level I, much more than the increase in computation time. Note that the handshake time did not increase linearly with the 55 760 bytes (108.6 %) increase of the SPHINCS$^+$-192f public keys and signatures. The TCP

Table 11: Average handshake times in ms for unilaterally authenticated post-quantum TLS experiments at NIST level III with 30.9 ms latency and 1000 Mbps bandwidth. Server and client timers are independent.

	Intermediate cert. as root			**With root certificate**		
Handle	**Client sent req.**	**Client recv. resp.**	**Server HS done**	**Client sent req.**	**Client recv. resp.**	**Server HS done**
errr	65.9	97.0	35.0	66.1	97.2	35.2
KDDD	64.0	95.1	33.1	94.6	125.8	63.7
KFFF	66.5	97.6	35.5	67.0	98.1	36.0
KDFF	64.3	95.4	33.3	65.0	96.1	34.0
KSfSfSf	145.6	176.7	114.6	178.5	209.7	147.6
KSsSsSs	215.3	246.3	184.4	248.1	279.2	217.2
KSsXX	223.3	254.3	192.3	231.9	262.9	201.0
KDXX	103.5	134.6	72.6	117.1	148.1	86.1
HDDD	64.0	95.2	33.1	94.6	125.8	63.7

Table 12: Average handshake times in ms for unilaterally authenticated post-quantum TLS experiments at NIST level III with 195.5 ms latency and 10 Mbps bandwidth. Server and client timers are independent.

	Intermediate cert. as root			**With root certificate**		
Handle	**Client sent req.**	**Client recv. resp.**	**Server HS done**	**Client sent req.**	**Client recv. resp.**	**Server HS done**
errr	397.1	593.1	201.3	397.7	593.7	201.8
KDDD	409.1	607.6	211.6	881.0	1077.0	674.2
KFFF	401.3	597.3	204.8	428.2	631.1	229.5
KDFF	403.2	599.9	206.5	413.6	613.3	215.4
KSfSfSf	2028.4	2510.0	1713.2	3427.2	3814.2	3147.7
KSsSsSs	1156.5	1490.4	938.0	1776.4	2044.8	1553.1
KSsXX	869.0	1083.5	666.6	876.9	1098.1	672.4
KDXX	456.1	659.4	249.7	495.2	703.7	282.2
HDDD	408.4	606.8	211.1	881.0	1077.0	674.2

congestion control algorithm increases the transmission window, and thus available bandwidth, as more packets get acknowledged. In the same experiment running over the 195.5 ms latency, 10 Mbps network, we see that the increase in handshake time in the same experiment is 87.1 %. This is much more in line with the increase in data that is transmitted, due to the lower connection bandwidth.

As the sizes of the handshakes go up, the benefits increase of choosing the algorithms for CA certificates such that the sizes are minimized. The KDFF instance is the best-performing post-quantum instance on the fast network when an intermediate certificate is transferred, balancing the computational speed of Dilithium3 with the smaller key sizes of Falcon-1024. On the low-bandwidth network, the size of Dilithium3 is too large to keep up with the KFFF instance, and the larger size slows down the handshake time more than the increase in Falcon's computational requirements: KDFF takes 17.8 ms (2.8 %) longer before the client response is received. Even the KDXX instance, which uses the slow-to-verify $\mathrm{XMSS}_s^{\mathrm{MT}}$-III scheme in the CA certificates, performs better than KDDD when intermediate CA certificates are used, showing the impact of the transmission overhead on low-bandwidth networks.

4.2 Mutually authenticated TLS 1.3

Communication requirements

In table 13, we show the communication requirements of our instantiations when using mutual authentication. As with the server authentication requirements in the unilaterally authenticated handshakes, the sizes of the public keys and signatures required for client authentication grow significantly. Already, no post-quantum instantiation has a total handshake size under 10 000 B when excluding intermediate CA certificates, and when including them, only the Falcon-1024-based KFFF-FF instance is just below 15 kB

Computational requirements

Table 14 shows the computational requirements for the server and the client. We see that using SPHINCS$^+$-192s for client authentication comes with a very significant computational cost, which now exceeds 100 ms for both the client and the server. We note that although TLS servers are often powerful computers that have stable power supplies, such computational overhead may have significant effects on the battery life of devices such as smartphones.

Handshake performance

Lastly, in tables 7 and 8, we show the handshake performance for mutually authenticated post-quantum TLS 1.3 handshakes at NIST security level III. For most instantiations, for the same reasons as in the unilaterally authenticated experiments, we do not see large differences between the results at the level I and level III security levels. Comparing the SPHINCS$^+$-192f-based instantiation to its level I instantiation when running on the 195.5 ms latency, 10 Mbps network shows that it takes 1809.6 ms (90.6 %) longer before the client receives its response than the level I instance using SPHINCS$^+$-128f (omitting intermediate CA certificates). The increase in size greatly affects the handshake performance on the low-bandwidth network.

On the high-bandwidth, low-latency network, the KSsXX-SsX instance got 170.2 ms (66.8 %) slower going from security level I to level III, while the unilaterally authenticated KSsXX instance got 67.7 ms (36.3 %) slower between the two security levels. On the high-latency, low-bandwidth network, this gap grows: the mutually authenticated instance was 552.4 ms (68.9 %) slower while the unilaterally authenticated instance was only 347.9 ms (47.3 %) slower. Somehow the combination of the increase in computational overhead and handshake size adds multiple round-trips worth of latency to the post-quantum TLS 1.3 handshake.

Table 13: Instantiations at NIST level III of mutually authenticated post-quantum TLS experiments and the sizes of the public-key cryptography elements transmitted in bytes.

Experiment handle	**Key** **exchange**	**Server** **authentication**	**Client** **authentication**	**Sum**	**Int. CA** **certificate**	**Sum**
Pre-quantum errr-rr	X25519 64	hs:RSA-2048 sig:RSA-2048 784	hs:RSA-2048 sig:RSA-2048 784	1 632	pk:RSA-2048 sig:RSA-2048 528	2 160
Primary KDDD-DD	Kyber-768 2272	hs:Dilithium3 sig:Dilithium3 8538	hs:Dilithium3 sig:Dilithium3 8538	19 348	pk:Dilithium3 sig:Dilithium3 5245	24 593
Falcon KFFF-FF	Kyber-768 2272	hs:Falcon-1024 sig:Falcon-1024 4353	hs:Falcon-1024 sig:Falcon-1024 4353	10 978	pk:Falcon-1024 sig:Falcon-1024 3073	14 051
Falcon offline KDFF-DF	Kyber-768 2272	hs:Dilithium3 sig:Falcon-1024 6525	hs:Dilithium3 sig:Falcon-1024 6525	15 322	pk:Falcon-1024 sig:Falcon-1024 3073	18 395
SPHINCS^+-f KSfSfSf-SfSf	Kyber-768 2272	hs:SPHINCS^+-192f sig:SPHINCS^+-192f 71 376	hs:SPHINCS^+-192f sig:SPHINCS^+-192f 71 376	145 024	pk:SPHINCS^+-192f sig:SPHINCS^+-192f 35 712	180 736
SPHINCS^+-s KSsSsSs-SsSs	Kyber-768 2272	hs:SPHINCS^+-192s sig:SPHINCS^+-192s 32 496	hs:SPHINCS^+-192s sig:SPHINCS^+-192s 32 496	67 264	pk:SPHINCS^+-192s sig:SPHINCS^+-192s 16 272	83 536
Hash-based signatures KSsXX-SsX	Kyber-768 2272	hs:SPHINCS^+-192s sig:$\text{XMSS}_\text{s}^\text{MT}$-III 18 123	hs:SPHINCS^+-192s sig:$\text{XMSS}_\text{s}^\text{MT}$-III 18 123	38 518	pk:$\text{XMSS}_\text{s}^\text{MT}$-III sig:$\text{XMSS}_\text{s}^\text{MT}$-III 1899	40 417
HBS-CA KDXX-DX	Kyber-768 2272	hs:Dilithium3 sig:$\text{XMSS}_\text{s}^\text{MT}$-III 7096	hs:Dilithium3 sig:$\text{XMSS}_\text{s}^\text{MT}$-III 7096	16 464	pk:$\text{XMSS}_\text{s}^\text{MT}$-III sig:$\text{XMSS}_\text{s}^\text{MT}$-III 1899	18 363
HQC HDDD-DD	HQC-192 13 548	hs:Dilithium3 sig:Dilithium3 8538	hs:Dilithium3 sig:Dilithium3 8538	30 624	pk:Dilithium3 sig:Dilithium3 5245	35 869

hs: certificate public key and handshake signature
pk: certificate public key sig: certificate signature

Table 14: Computation time in ms for asymmetric cryptography at NIST level III for each of the mutually authenticated post-quantum TLS instantiations at the client and server.

Handle	**Intermediate cert. as root**			**With intermediate CA cert.**		
	Client	Server	Sum	Client	Server	Sum
errr-rr	0.660	0.661	1.321	0.676	0.661	1.337
KDDD-DD	1.006	0.992	1.998	1.087	0.992	2.079
KFFF-FF	1.171	1.157	2.328	1.343	1.157	2.500
KDFF-DF	1.097	1.083	2.180	1.269	1.083	2.352
KSfSfSf-SfSf	7.075	7.061	14.136	7.475	7.061	14.536
KSsSsSs-SsSs	119.169	119.155	238.324	119.304	119.155	238.459
KSsXX-SsX	130.933	130.919	261.852	142.832	130.919	273.751
KDXX-DX	12.824	12.810	25.634	24.723	12.810	37.533
HDDD-DD	1.993	1.425	3.418	2.074	1.425	3.499

Table 15: Average handshake times in ms for mutually authenticated post-quantum TLS experiments at NIST level III with 30.9 ms latency and 1000 Mbps bandwidth. Server and client timers are independent.

	Intermediate cert. as root			**With root certificate**		
Handle	**Client sent req.**	**Client recv. resp.**	**Server HS done**	**Client sent req.**	**Client recv. resp.**	**Server HS done**
errr-rr	68.9	100.3	38.3	68.9	100.4	38.3
KDDD-DD	64.9	96.6	34.6	95.4	127.2	65.1
KFFF-FF	69.1	101.4	39.4	69.8	102.2	40.1
KDFF-DF	65.1	97.2	35.2	65.7	97.8	35.8
KSfSfSf-SfSf	166.1	261.4	199.3	198.1	293.5	231.3
KSsSsSs-SsSs	361.6	424.3	362.4	396.4	459.2	397.2
KSsXX-SsX	362.7	425.0	363.1	364.2	426.6	364.6
KDXX-DX	105.6	176.0	114.0	117.7	181.5	119.5
HDDD-DD	64.9	96.7	34.6	95.4	127.2	65.1

Table 16: Average handshake times in ms for mutually authenticated post-quantum TLS experiments at NIST level III with 195.5 ms latency and 10 Mbps bandwidth. Server and client timers are independent.

	Intermediate cert. as root			**With root certificate**		
Handle	**Client sent req.**	**Client recv. resp.**	**Server HS done**	**Client sent req.**	**Client recv. resp.**	**Server HS done**
errr-rr	399.9	597.2	205.2	401.0	598.4	206.1
KDDD-DD	409.2	615.6	219.0	795.3	1067.3	559.5
KFFF-FF	404.1	605.1	212.6	410.1	612.3	217.0
KDFF-DF	404.6	607.5	213.3	411.2	616.5	217.6
KSfSfSf-SfSf	2198.3	3806.7	2982.4	3571.3	5450.1	4575.4
KSsSsSs-SsSs	1130.8	1682.2	1178.4	1466.6	2050.8	1489.1
KSsXX-SsX	918.9	1354.2	934.7	926.3	1359.7	939.4
KDXX-DX	459.4	707.9	301.5	496.0	744.6	325.5
HDDD-DD	408.9	615.4	218.8	788.7	1055.5	547.4

5 Instantiation and results at NIST level V

In this section, we examine the characteristics of post-quantum TLS instances at NIST PQC security level V. This is the most conservative security level and should offer comparable security to AES-256. This security level is required by the United States National Security Agency (NSA)'s Commercial National Security Algorithm Suite 2.0 [29]. The French national cybersecurity agency agence nationale de la sécurité des systèmes d'information (ANSSI) also recommends security level V [3]. As the most conservative parameter sets, these are generally the largest and slowest-running algorithms to instantiate post-quantum TLS 1.3 with. As such, it will be challenging to instantiate TLS with these algorithms without affecting performance significantly.

5.1 Unilaterally authenticated TLS 1.3

Communication requirements

In table 17, we show the communication sizes of our choices of instantiations. This level has the most conservative security guarantees, and as such the largest key sizes and computational requirements. If we compare the sums of the public-key cryptography elements that need to be transmitted, we see that even when leaving out the intermediate CA certificates, all but the KFFF instance exceed 10 kB. When we compare the sums including the intermediate CA certificate, all but KFFF and KDFF exceed 15 kB. SPHINCS$^+$-256f even exceeds 150 kB.

Computational requirements

When we compare the computation time that is necessary for the public-key cryptography operations at level V in table 18 with the level III requirements, it seems that the increase is less than going from level I to level III. Although HQC-256 requires 0.571 ms (38.4 %) more time than HQC-192, Kyber-1024 only requires 0.052 ms (76.5 %) more than Kyber-768. Dilithium5's signing time is almost identical to Dilithium3's. Falcon-1024, which we used in our level III experiments, already has security level V. The hash-based schemes SPHINCS$^+$ and XMSS$_s^{MT}$use the same hash primitives for level III and level V, but they truncate the output of the hash function to 192 bytes at level III, while obtaining 256 bytes at level V. SPHINCS$^+$-256s additionally has a slightly differently shaped tree, which optimizes differently for signing and verification time. The performance of these schemes is thus very similar at level III and V, with SPHINCS$^+$-256s signing being slightly faster than SPHINCS$^+$-192s.

Handshake performance

In tables 19 and 20, we show the handshake times for the instantiations at NIST security level V. In the results for the instances run on the 30.9 ms latency, 1000 Mbps network, we see that the experiments that are under the `initcwnd` size limit still complete in roughly the same amount of time, at multiples of the round trip latency. However, when including the intermediate certificate we see that all instances but KFFF and KDFF suffer at least an extra round-trip worth of handshake time due to exceeding the limit of the initial congestion window. Compared to the pre-quantum experiment, KDDD now requires 29.8 ms (30.6 %) more time if an intermediate certificate needs to be transmitted. In the low-bandwidth, 195.5 ms latency experiment KDDD even needs 488.1 ms (82.2 %) more time. Even the KFFF instance sees a 34.6 ms (5.8 %) increase in time when an intermediate certificate is included, again illustrating the impact of the larger certificates on low-bandwidth connections. If Falcon-1024 is not an option for the online handshake signatures, using it for CA certificates in the KDFF instantiation can mitigate the performance penalty compared to KDDD a bit, by improving the performance to a slowdown of 25.8 ms (4.3 %) compared to the pre-quantum instance errr.

The performance of the instances using SPHINCS$^+$-256 variants is not too much slower than SPHINCS$^+$-192 on the fast network, even when the large intermediate CA certificates are transmitted. The instance

Table 17: Instantiations at NIST level V of unilaterally authenticated post-quantum TLS handshakes and the sizes of the public-key cryptography elements in bytes.

		Leaf certificate			**Int. CA certificate**			**Offline**
Experiment handle	**Key Exchange** pk+ct	**Handshake auth.** pk+sig	**Int. CA signature** sig	**Sum**	**Int. CA public key** pk	**Root CA signature** sig	**Sum**	**Root CA public key** pk
Pre-quantum errr	X25519 64	RSA-2048 528	RSA-2048 256	848	RSA-2048 272	RSA-2048 256	1 376	RSA-2048 272
Primary KDDD	Kyber-1024 3136	Dilithium5 7187	Dilithium5 4595	14 918	Dilithium5 2592	Dilithium5 4595	22 105	Dilithium5 2592
Falcon KFFF	Kyber-1024 3136	Falcon-1024 3073	Falcon-1024 1280	7 489	Falcon-1024 1793	Falcon-1024 1280	10 562	Falcon-1024 1793
Falcon offline KDFF	Kyber-1024 3136	Dilithium5 7187	Falcon-1024 1280	11 603	Falcon-1024 1793	Falcon-1024 1280	14 676	Falcon-1024 1793
SPHINCS$^+$-f KSfSfSf	Kyber-1024 3136	SPHINCS$^+$-256f 49 920	SPHINCS$^+$-256f 49 856	102 912	SPHINCS$^+$-256f 64	SPHINCS$^+$-256f 49 856	152 832	SPHINCS$^+$-256f 64
SPHINCS$^+$-s KSsSsSs	Kyber-1024 3136	SPHINCS$^+$-256s 29 856	SPHINCS$^+$-256s 29 792	62 784	SPHINCS$^+$-256s 64	SPHINCS$^+$-256s 29 792	92 640	SPHINCS$^+$-256s 64
Hash-based signatures KSsXX	Kyber-1024 3136	SPHINCS$^+$-256s 29 856	$\text{XMSS}^{\text{MT}}_{\text{s}}$-V 2979	35 971	$\text{XMSS}^{\text{MT}}_{\text{s}}$-V 64	$\text{XMSS}^{\text{MT}}_{\text{s}}$-V 2979	39 014	$\text{XMSS}^{\text{MT}}_{\text{s}}$-V 64
HBS-CA KDXX	Kyber-1024 3136	Dilithium5 7187	$\text{XMSS}^{\text{MT}}_{\text{s}}$-V 2979	13 302	$\text{XMSS}^{\text{MT}}_{\text{s}}$-V 64	$\text{XMSS}^{\text{MT}}_{\text{s}}$-V 2979	16 345	$\text{XMSS}^{\text{MT}}_{\text{s}}$-V 64
HQC HDDD	HQC-256 21 714	Dilithium5 7187	Dilithium5 4595	33 496	Dilithium5 2592	Dilithium5 4595	40 683	Dilithium5 2592

Table 18: Computation time in ms for asymmetric cryptography at NIST level V for each of the unilaterally authenticated post-quantum TLS instantiations at the client and server.

Handle	**Intermediate cert. as root**			**With intermediate CA cert.**		
	Client	Server	Sum	Client	Server	Sum
errr	0.134	0.629	0.763	0.150	0.629	0.779
KDDD	0.398	0.633	1.031	0.559	0.633	1.192
KFFF	0.420	0.830	1.250	0.592	0.830	1.422
KDFF	0.409	0.633	1.042	0.581	0.633	1.214
KSfSfSf	0.870	11.406	12.276	1.267	11.406	12.673
KSsSsSs	0.444	104.611	105.055	0.628	104.611	105.239
KSsXX	11.295	104.611	115.906	22.330	104.611	126.941
KDXX	11.272	0.633	11.905	22.307	0.633	22.940
HDDD	1.753	1.217	2.970	1.914	1.217	3.131

Table 19: Average handshake times in ms for unilaterally authenticated post-quantum TLS experiments at NIST level V with 30.9 ms latency and 1000 Mbps bandwidth. Server and client timers are independent.

	Intermediate cert. as root			**With root certificate**		
Handle	**Client sent req.**	**Client recv. resp.**	**Server HS done**	**Client sent req.**	**Client recv. resp.**	**Server HS done**
errr	65.9	97.0	35.0	66.1	97.2	35.2
KDDD	64.5	95.6	33.5	95.9	127.0	65.0
KFFF	66.4	97.5	35.4	67.1	98.2	36.2
KDFF	64.6	95.7	33.6	65.2	96.4	34.3
KSfSfSf	169.8	200.9	138.8	198.2	229.4	167.2
KSsSsSs	238.9	270.0	208.0	247.0	278.1	216.1
KSsXX	215.1	246.2	184.2	230.6	261.7	199.7
KDXX	110.9	142.0	80.0	121.1	152.2	90.2
HDDD	64.5	95.6	33.5	95.9	127.1	65.0

Table 20: Average handshake times in ms for unilaterally authenticated post-quantum TLS experiments at NIST level V with 195.5 ms latency and 10 Mbps bandwidth. Server and client timers are independent.

	Intermediate cert. as root			**With root certificate**		
Handle	**Client sent req.**	**Client recv. resp.**	**Server HS done**	**Client sent req.**	**Client recv. resp.**	**Server HS done**
errr	397.1	593.1	201.3	397.7	593.7	201.8
KDDD	416.4	616.5	216.6	885.8	1081.8	689.3
KFFF	401.3	597.3	204.8	426.8	628.3	227.5
KDFF	407.2	604.8	209.9	418.7	619.5	218.5
KSfSfSf	2748.2	3589.5	2482.5	4321.3	5100.0	3707.2
KSsSsSs	2047.6	2397.9	1819.3	2707.4	3445.2	2421.1
KSsXX	1172.1	1515.5	958.9	1205.3	1584.0	980.5
KDXX	474.2	680.6	265.8	879.9	1076.6	671.7
HDDD	418.4	618.9	218.0	885.7	1081.7	689.2

based on SPHINCS$^+$-256f (KSfSfSf) is 19.7 ms (9.4 %) slower at level V than at level III, much less than the 42 608 bytes (39.8 %) increase in signature size. However, it is still 102.3 ms (80.6 %) slower than the KDDD instantiation, and 132.1 ms (135.9 %) slower than using pre-quantum cryptography. On the low-bandwidth network, the additional bandwidth requirements very significantly slow down the TLS handshakes. Here, KSfSfSf is 1285.7 ms (33.7 %) slower at level V compared to level III. The size-optimized instance that uses SPHINCS$^+$-256s, KSsSsSs, is 1654.8 ms (32.4 %) faster than KSfSfSf, easily making up for the additional computational requirements, but still 4018.2 ms (371.4 %) slower than KDDD. Again, we see that using XMSS$_s^{MT}$-V instead of SPHINCS$^+$-256s leads to a significant increase in handshake performance: the KSsXX instance takes 1861.1 ms (54.0 %) less time before the client receives a response.

Table 21: Instantiations at NIST level V of mutually authenticated post-quantum TLS experiments and the sizes of the public-key cryptography elements transmitted in bytes.

Experiment handle	**Key exchange**	**Server authentication**	**Client authentication**	**Sum**	**Int. CA certificate**	**Sum**
Pre-quantum errr-rr	X25519 64	hs:RSA-2048 sig:RSA-2048 784	hs:RSA-2048 sig:RSA-2048 784	1 632	pk:RSA-2048 sig:RSA-2048 528	2 160
Primary KDDD-DD	Kyber-1024 3136	hs:Dilithium5 sig:Dilithium5 11 782	hs:Dilithium5 sig:Dilithium5 11 782	26 700	pk:Dilithium5 sig:Dilithium5 7187	33 887
Falcon KFFF-FF	Kyber-1024 3136	hs:Falcon-1024 sig:Falcon-1024 4353	hs:Falcon-1024 sig:Falcon-1024 4353	11 842	pk:Falcon-1024 sig:Falcon-1024 3073	14 915
Falcon offline KDFF-DF	Kyber-1024 3136	hs:Dilithium5 sig:Falcon-1024 8467	hs:Dilithium5 sig:Falcon-1024 8467	20 070	pk:Falcon-1024 sig:Falcon-1024 3073	23 143
SPHINCS$^+$-f KSfSfSf-SfSf	Kyber-1024 3136	hs:SPHINCS$^+$-256f sig:SPHINCS$^+$-256f 99 776	hs:SPHINCS$^+$-256f sig:SPHINCS$^+$-256f 99 776	202 688	pk:SPHINCS$^+$-256f sig:SPHINCS$^+$-256f 49 920	252 608
SPHINCS$^+$-s KSsSsSs-SsSs	Kyber-1024 3136	hs:SPHINCS$^+$-256s sig:SPHINCS$^+$-256s 59 648	hs:SPHINCS$^+$-256s sig:SPHINCS$^+$-256s 59 648	122 432	pk:SPHINCS$^+$-256s sig:SPHINCS$^+$-256s 29 856	152 288
Hash-based signatures KSsXX-SsX	Kyber-1024 3136	hs:SPHINCS$^+$-256s sig:XMSS$_s^{MT}$-V 32 835	hs:SPHINCS$^+$-256s sig:XMSS$_s^{MT}$-V 32 835	68 806	pk:XMSS$_s^{MT}$-V sig:XMSS$_s^{MT}$-V 3043	71 849
HBS-CA KDXX-DX	Kyber-1024 3136	hs:Dilithium5 sig:XMSS$_s^{MT}$-V 10 166	hs:Dilithium5 sig:XMSS$_s^{MT}$-V 10 166	23 468	pk:XMSS$_s^{MT}$-V sig:XMSS$_s^{MT}$-V 3043	26 511
HQC HDDD-DD	HQC-256 21 714	hs:Dilithium5 sig:Dilithium5 11 782	hs:Dilithium5 sig:Dilithium5 11 782	45 278	pk:Dilithium5 sig:Dilithium5 7187	52 465

hs: certificate public key and handshake signature
pk: certificate public key sig: certificate signature

5.2 Mutually authenticated TLS 1.3

Communication requirements

Table 21 shows how we instantiate mutually authenticated post-quantum TLS 1.3 at level V and the sizes of the public keys and signatures. Now, all instantiations exceed 10 kB even when omitting intermediate CA certificates. The Kyber-1024 and Falcon-1024-based KFFF-FF instance is only marginally larger at level V than at level III, however, as we already used level-V secure Falcon-1024 in the level III instantiation.

Computational requirements

Table 22 shows the computational requirements of the cryptography in these instantiations at level V. We observe the same things as for the unilaterally authenticated handshakes, except the client now also needs to produce a signature while the server has to verify this signature and the signature in the client certificate.

Table 22: Computation time in ms for asymmetric cryptography at NIST level V for each of the mutually authenticated post-quantum TLS instantiations at the client and server.

Handle	**Intermediate cert. as root**			**With intermediate CA cert.**		
	Client	Server	Sum	Client	Server	Sum
errr-rr	0.660	0.661	1.321	0.676	0.661	1.337
KDDD-DD	0.987	0.955	1.942	1.148	0.955	2.103
KFFF-FF	1.206	1.174	2.380	1.378	1.174	2.552
KDFF-DF	0.998	0.966	1.964	1.170	0.966	2.136
KSfSfSf-SfSf	12.232	12.200	24.432	12.629	12.200	24.829
KSsSsSs-SsSs	105.011	104.979	209.990	105.195	104.979	210.174
KSsXX-SsX	115.862	115.830	231.692	126.897	115.830	242.727
KDXX-DX	11.861	11.829	23.690	22.896	11.829	34.725
HDDD-DD	2.342	1.539	3.881	2.503	1.539	4.042

Handshake performance

In tables 23 and 24, we show the handshake performance. In both tables, the results are in line with what we have seen for the unilaterally authenticated handshakes and the level III experiments.

Table 23: Average handshake times in ms for mutually authenticated post-quantum TLS experiments at NIST level V with 30.9 ms latency and 1000 Mbps bandwidth. Server and client timers are independent.

	Intermediate cert. as root			**With root certificate**		
Handle	**Client sent req.**	**Client recv. resp.**	**Server HS done**	**Client sent req.**	**Client recv. resp.**	**Server HS done**
errr-rr	68.9	100.3	38.3	68.9	100.4	38.3
KDDD-DD	65.4	97.5	35.5	96.9	129.0	66.9
KFFF-FF	69.0	101.4	39.3	69.8	102.1	40.1
KDFF-DF	65.5	97.7	35.7	66.2	98.5	36.4
KSfSfSf-SfSf	230.9	333.7	271.5	237.3	340.6	278.4
KSsSsSs-SsSs	387.5	481.2	419.2	387.9	481.6	419.6
KSsXX-SsX	346.2	408.8	346.9	346.5	409.1	347.1
KDXX-DX	107.0	184.3	122.3	122.3	192.1	130.1
HDDD-DD	65.5	97.6	35.5	96.8	128.9	66.8

Table 24: Average handshake times in ms for mutually authenticated post-quantum TLS experiments at NIST level V with 195.5 ms latency and 10 Mbps bandwidth. Server and client timers are independent.

	Intermediate cert. as root			**With root certificate**		
Handle	**Client sent req.**	**Client recv. resp.**	**Server HS done**	**Client sent req.**	**Client recv. resp.**	**Server HS done**
errr-rr	399.9	597.2	205.2	401.0	598.4	206.1
KDDD-DD	426.0	641.7	233.3	801.1	1042.0	601.9
KFFF-FF	404.0	605.1	212.6	409.9	612.1	216.9
KDFF-DF	410.2	617.9	220.5	417.9	626.8	223.7
KSfSfSf-SfSf	3117.7	4918.7	3790.7	4577.6	7095.0	5936.6
KSsSsSs-SsSs	1504.9	2422.4	1834.0	2255.4	3630.6	2887.9
KSsXX-SsX	1198.6	1809.2	1282.4	1312.4	1973.6	1442.1
KDXX-DX	483.9	753.1	328.9	628.4	895.1	478.5
HDDD-DD	427.2	642.9	234.2	785.5	1032.3	590.9

6 Discussion

Summarizing the results for our experiments across NIST security level I, III, and V, we see that Kyber and Dilithium offer reasonable performance. The size of Dilithium does affect the handshake times significantly at levels III and V, due to these instances exceeding the initial congestion window size. Falcon offers comparable or better performance, as the AVX2-accelerated implementation is highly performant and its public keys and signatures are the smallest. If using Falcon for online signatures is not an option due to its requirement of constant-time 64-bit floating-point arithmetic for signing, using it with Dilithium for online signatures as in our KDFF instances is very attractive, although this does increase the size of the codebase. Using SPHINCS$^+$ does not seem very attractive for TLS 1.3: both in computation time and size of signatures, the scheme very significantly affects the handshake performance. If any application greatly prefers using a hash-based signature scheme, it seems using alternatives to SPHINCS$^+$ for CA certificates can offer very large performance improvements, especially at higher security levels.

Our conclusions are also reflected by figure 1: all of the instantiations are mostly gathered together around the 3-RTT line. Only the instances that use large certificates or that use slower algorithms such as XMSS$_s^{MT}$ move away from this line. In the bottom plot, we compare all of the instantiations using SPHINCS$^+$: they are so large and comparatively slow that the top graph fits in the lower left corner of the bottom graph.

Our experiment is not the first to look at the performance of post-quantum TLS. Comparing our results with [30, 38, 39, 40, 41], we see that we arrive at similar results. For high-bandwidth connections, the increase in computation time is very moderate for any scheme that stays under the `initcwnd` limit on the number of MSS that can be sent before receiving a TCP acknowledgment from the recipient. As Sikideris, Kampanakis, and Devetsikiotis first observed in [39], we see that combining two different algorithms for *online* handshake authentication and *offline* CA certificates can greatly influence connection establishment times.

As Westerbaan [41] highlighted, there are many more signatures in a typical TLS handshake on the web than we included in our experiments. The Google Chrome and Safari web browsers require at least two certificate transparency (CT) [25] proofs to be included for any certificate [4, 12]. This means that in every certificate, there are two additional signatures from CT logs. The online certificate status protocol (OCSP) [34], which allows a server to show that its certificate is not revoked, also consists of a signed statement from the certificate authority. CT and OCSP thus add another three offline signatures to the typical TLS handshake traffic. If all of the signature algorithms in the TLS handshake are instantiated with Dilithium2, this adds up to 17 144 bytes of public keys and signatures that are sent in the `ServerCertificate` message. This easily exceeds a 10 MSS initial congestion window size. This means using different algorithms for online and offline signatures and/or standardizing mechanisms that allow removing or suppressing of some of the signatures [22] may be vital to the performance of TLS on the web.

The initial congestion window size default of 10 MSS is a fairly recent development, first suggested by Google [16]. This suggestion was implemented by Linux in 2011, before being standardized by the IETF in RFC 6928 in 2013 [13]. Before, the recommendation was a congestion window "between 2 and 4 segments". It has been argued that the congestion window can be increased to allow for the larger sizes of post-quantum cryptographic algorithms, e.g. in [6]. Indeed, many content delivery networks already use much larger congestion windows [33]. Increasing the initial congestion window across the internet may be a valid strategy for handling the increase in TLS handshake traffic. However, too large values could result in congestion and packet loss. RFC 6928 [13, Appendix A] additionally highlights that raising the congestion window may have adverse effects on internet connection speeds in the developing world. Evaluating what would be the best value for `initcwnd` and the impact of changing the value on the congestion control behavior of TCP across the internet is beyond the scope of this thesis, however, and we leave this for other work.

Finally, we have not explored the computational load of our instantiations. The algorithms have very different computational characteristics. If an algorithm has very high computational requirements, this

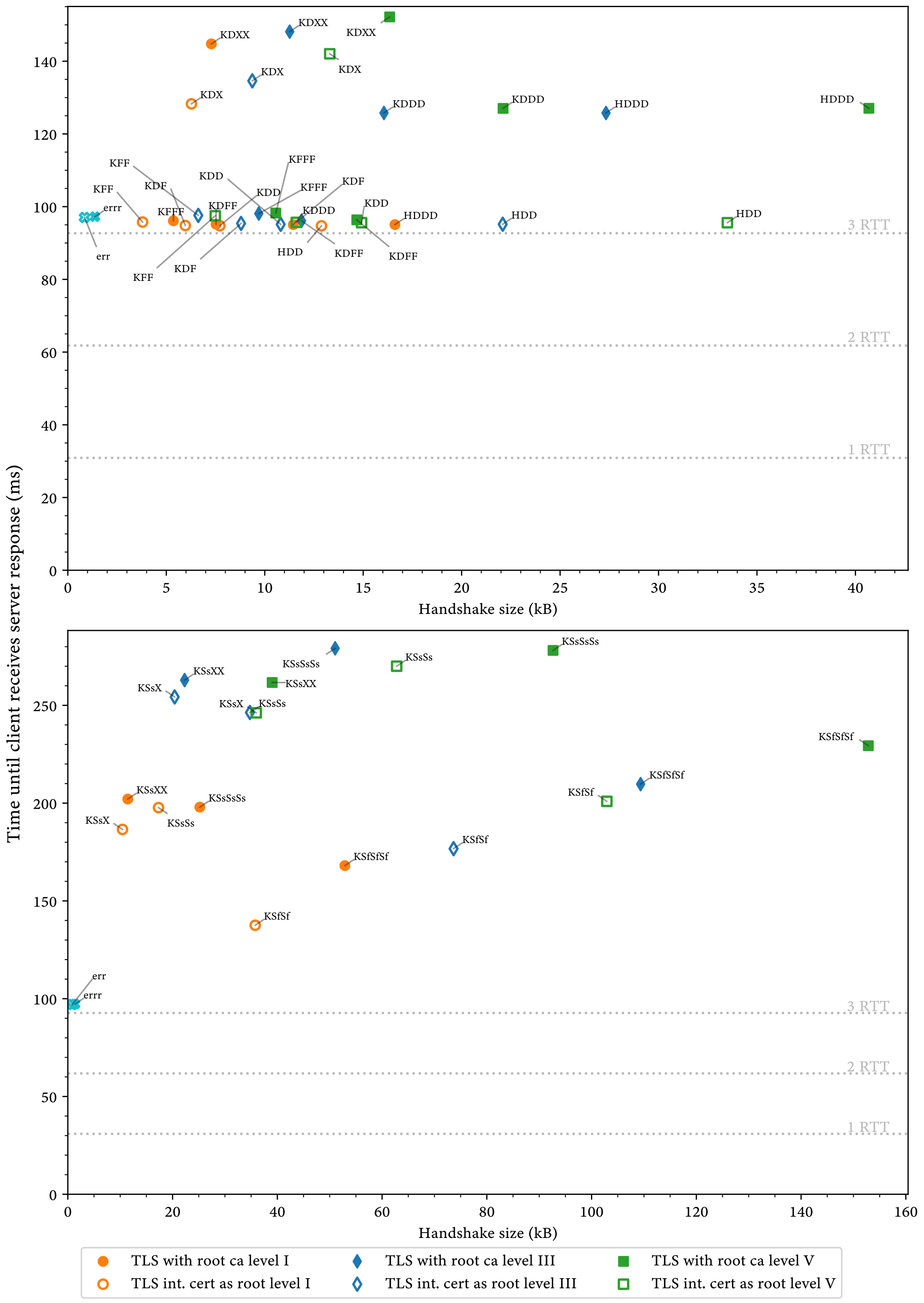

Figure 1: Handshake timings of post-quantum TLS experiments

may result in a limitation on the number of connections that a server may be able to support. Sikideris, Kampanakis, and Devetsikiotis examined this for unilaterally authenticated TLS 1.3 using round-2 schemes in [39] and showed that a server running Dilithium may be able to support more connections than a server running Falcon. In our experiments, the server was also exactly as computationally powerful as the client: after all, they ran on the same machine. Often, however, this is not the case: clients may, for example, have battery life concerns, such as smartphones or laptops. Further examining these issues in post-quantum TLS remains for future work. Another example of asymmetry between clients and servers is the case of the performance of post-quantum TLS on microcontrollers, which are much less powerful and may have low-bandwidth links. We will discuss this issue in chapter 16.

7 Appendix: XMSS at different NIST security levels

The security of XMSS parameter sets specified in RFC 8391 [18] reach NIST security level V (equivalent to AES-256) and above. This high level of security has only a very minor impact on computational performance, but it does have a significant impact on signature size. The NIST standard [14] also considers parameter sets targeting security level III (equivalent to AES-192); the simple modification is to truncate all hashes to 192 bits. This is possible because the security of XMSS and its multi-tree variant XMSS^{MT} is guaranteed by a tight reduction from second-preimage resistance [8, 21]; collisions in the underlying hash function do not affect the security of XMSS. We straightforwardly extend XMSS to parameter sets targeting NIST security levels I and III. For level I, hashes are simply truncated to 128 bits; we obtain this by using SHAKE-128 [28] with 128 bits of output. For level III, we use SHAKE-256 truncated to 192 bits of output. SPHINCS^{+} similarly constructs its level-I and level-III parameter sets. Aside from minor details, XMSS can be seen as a "sub-step" of SPHINCS^{+}; see [19]. To complete our set of parameters, we also specify a variant at NIST security level V, which obtains 256 bits from SHAKE-256.

We define $\text{XMSS}_{\text{s}}^{\text{MT}}$ as an instantiation of XMSS^{MT} using two trees of height 12 each, i.e., a total tree height of 24, which limits the maximum number of signatures per public key to $2^{24} \approx 16.7$ million. Increasing this maximum number of signatures to, for example, $2^{30} \approx 1$ billion increases signature size by only 96 bytes and has negligible impact on verification speed. It does have an impact on key-generation speed and signing latency, but as mentioned in section 13.6.3, the latency of signing is not very relevant when used by certificate authorities as in our benchmarks.

Multi-tree XMSS is particularly well-suited for efficient batch signing. The idea is to compute one whole tree (of height h/d) on the lowest level and use it on-the-fly to sign $2^{h/d}$ messages. The computational effort per signature is then essentially reduced to one WOTS^{+} key-pair generation.

We set the Winternitz parameter in $\text{XMSS}_{\text{s}}^{\text{MT}}$ to $w = 256$ to optimize for signature size. Changing to the more common $w = 16$ would increase the signature size by about a factor of 2 and speed up verification by about a factor of 8.

References

[1] Gorjan Alagic, Daniel Apon, David Cooper, Quynh Dang, Thinh Dang, John Kelsey, Jacob Lichtinger, Carl Miller, Dustin Moody, Rene Peralta, Ray Perlner, Angela Robinson, Daniel Smith-Tone, and Yi-Kai Liu. *Status Report on the Third Round of the NIST Post-Quantum Cryptography Standardization Process*. Tech. rep. NISTIR 8413. Updated version. National Institute of Standards and Technology, Sept. 26, 2022. DOI: 10.6028/NIST.IR.8413-upd1 (cit. on p. 3).

[2] Martin R. Albrecht, Daniel J. Bernstein, Tung Chou, Carlos Cid, Jan Gilcher, Tanja Lange, Varun Maram, Ingo von Maurich, Rafael Misoczki, Ruben Niederhagen, Kenneth G. Paterson, Edoardo Persichetti, Christiane Peters, Peter Schwabe, Nicolas Sendrier, Jakub Szefer, Cen Jung Tjhai, Martin Tomlinson, and Wen Wang. *Classic McEliece*. Tech. rep. National Institute of Standards and Technology, 2022. URL: https://csrc.nist.gov/projects/post-quantum-cryptography/round-4-submissions, archived at https://web.archive.org/web/20230501144806/https://csrc.nist.gov/Projects/post-quantum-cryptography/round-4-submissions on May 1, 2023 (cit. on p. 5).

[3] *ANSSI views on the post-quantum cryptography transition*. Agence nationale de la sécurité des systèmes d'information. Jan. 4, 2022. URL: https://www.ssi.gouv.fr/en/publication/anssi-views-on-the-post-quantum-cryptography-transition/ (visited on Mar. 3, 2023), archived at https://web.archive.org/web/20230505121357/https://www.ssi.gouv.fr/en/publication/anssi-views-on-the-post-quantum-cryptography-transition/ on May 5, 2023 (cit. on p. 21).

[4] *Apple's Certificate Transparency policy*. Apple. Mar. 11, 2021. URL: https://support.apple.com/en-gb/HT205280 (visited on Oct. 6, 2022), archived at https://web.archive.org/web/20230329153011/https://support.apple.com/en-gb/HT205280 on Mar. 29, 2023 (cit. on p. 27).

[5] Nicolas Aragon, Paulo Barreto, Slim Bettaieb, Loic Bidoux, Olivier Blazy, Jean-Christophe Deneuville, Phillipe Gaborit, Shay Gueron, Tim Guneysu, Carlos Aguilar Melchor, Rafael Misoczki, Edoardo Persichetti, Nicolas Sendrier, Jean-Pierre Tillich, Gilles Zémor, Valentin Vasseur, Santosh Ghosh, and Jan Richter-Brokmann. *BIKE*. Tech. rep. National Institute of Standards and Technology, 2022. URL: https://csrc.nist.gov/Projects/post-quantum-cryptography/round-4-submissions, archived at https://web.archive.org/web/20230501144806/https://csrc.nist.gov/Projects/post-quantum-cryptography/round-4-submissions on May 1, 2023 (cit. on p. 5).

[6] Daniel J. Bernstein. *Cryptographic competitions*. Cryptology ePrint Archive, Report 2020/1608. Dec. 27, 2020. IACR ePrint: ia.cr/2020/1608 (cit. on p. 27).

[7] Daniel J. Bernstein. "Curve25519: New Diffie-Hellman Speed Records." In: *PKC 2006: 9th International Conference on Theory and Practice of Public Key Cryptography*. Ed. by Moti Yung, Yevgeniy Dodis, Aggelos Kiayias, and Tal Malkin. Vol. 3958. Lecture Notes in Computer Science. New York, NY, USA: Springer, Heidelberg, Germany, Apr. 24–26, 2006, pp. 207–228. DOI: 10.1007/11745853_14 (cit. on p. 4).

[8] Daniel J. Bernstein and Andreas Hülsing. "Decisional Second-Preimage Resistance: When Does SPR Imply PRE?" In: *Advances in Cryptology – ASIACRYPT 2019, Part III*. Ed. by Steven D. Galbraith and Shiho Moriai. Vol. 11923. Lecture Notes in Computer Science. Kobe, Japan: Springer, Heidelberg, Germany, Dec. 8–12, 2019, pp. 33–62. DOI: 10.1007/978-3-030-34618-8_2. IACR ePrint: ia.cr/2019/492 (cit. on p. 29).

[9] Eric W. Biederman and Nicolas Dichtel. *ip-netns(8)*. man ip netns. Jan. 2013. URL: https://man7.org/linux/man-pages/man8/ip-netns.8.html (cit. on p. 2).

[10] Moritz Birghan and Thyla van der Merwe. "A Client-Side Seat to TLS Deployment." In: *2022 SecWeb Workshop on Designing Security for the Web (2022 IEEE S&P Workshops)*. Genoa, Italy, May 26, 2022, pp. 13–19. DOI: 10.1109/SPW54247.2022.9833861. URL: https://secweb.work/papers/2022/birghan2022clienttls.pdf (cit. on p. 5).

[11] Ethan Blanton, Vern Paxson, and Mark Allman. *TCP Congestion Control*. RFC 5681. Sept. 2009. DOI: 10.17487/RFC5681 (cit. on p. 9).

[12] *Chrome Certificate Transparency Policy*. Google. Mar. 18, 2022. URL: https://github.com/GoogleChrome/CertificateTransparency/blob/827e20c9e5f6486ae18cfe99e25cd1e67fde1e15/ct_policy.md (visited on Oct. 6, 2022), archived at https://web.archive.org/web/20230509140101/https://github.com/GoogleChrome/CertificateTransparency/blob/827e20c9e5f6486ae18cfe99e25cd1e67fde1e15/ct_policy.md on May 9, 2023 (cit. on p. 27).

[13] Jerry Chu, Nandita Dukkipati, Yuchung Cheng, and Matt Mathis. *Increasing TCP's Initial Window*. RFC 6928. Apr. 2013. DOI: 10.17487/RFC6928 (cit. on p. 27).

[14] David Cooper, Daniel Apon, Quynh Dang, Michael Davidson, Morris Dworkin, and Carl Miller. *SP800-208: Recommendation for Stateful Hash-Based Signature Schemes*. DOI: 10.6028/NIST.SP.800-208 (cit. on pp. 3, 29).

[15] *Dilithium*. CRYSTALS-Dilithium team. Feb. 16, 2021. URL: https://pq-crystals.org/dilithium/ (visited on Mar. 6, 2023), archived at https://web.archive.org/web/20230323084032/https://pq-crystals.org/dilithium/ on Mar. 23, 2023 (cit. on p. 14).

[16] Nandita Dukkipati, Tiziana Refice, Yuchung Cheng, Jerry Chu, Tom Herbert, Amit Agarwal, Arvind Jain, and Natalia Sutin. "An Argument for Increasing TCP's Initial Congestion Window." In: *ACM SIGCOMM Computer Communication Review* 40.3 (June 22, 2010), pp. 26–33. DOI: 10.1145/1823844.1823848. URL: https://research.google/pubs/pub36640/ (cit. on p. 27).

[17] Stephen Hemminger, Fabio Ludovici, and Hagen Paul Pfeiffer. *tc-netem(8)*. man tc netem. Linux foundation. Nov. 2011. URL: https://man7.org/linux/man-pages/man8/tc-netem.8.html (cit. on p. 2).

[18] Andreas Huelsing, Denis Butin, Stefan-Lukas Gazdag, Joost Rijneveld, and Aziz Mohaisen. *XMSS: eXtended Merkle Signature Scheme*. RFC 8391. May 2018. DOI: 10.17487/RFC8391 (cit. on pp. 3, 29).

[19] Andreas Hülsing. *Public Comments on Draft SP 800-208*. https://csrc.nist.gov/CSRC/media/Publications/sp/800-208/draft/documents/sp800-208-draft-comments-received.pdf, page 7. 2020, archived at https://web.archive.org/web/20220605052334/https://csrc.nist.gov/CSRC/media/Publications/sp/800-208/draft/documents/sp800-208-draft-comments-received.pdf on June 5, 2022 (cit. on p. 29).

[20] Andreas Hülsing, Daniel J. Bernstein, Christoph Dobraunig, Maria Eichlseder, Scott Fluhrer, Stefan-Lukas Gazdag, Panos Kampanakis, Stefan Kölbl, Tanja Lange, Martin M. Lauridsen, Florian Mendel, Ruben Niederhagen, Christian Rechberger, Joost Rijneveld, Peter Schwabe, Jean-Philippe Aumasson, Bas Westerbaan, and Ward Beullens. *SPHINCS$^+$*. Tech. rep. National Institute of Standards and Technology, 2022. URL: https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022, archived at https://web.archive.org/web/20230429160244/https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022 on Apr. 29, 2023 (cit. on p. 4).

[21] Andreas Hülsing, Joost Rijneveld, and Fang Song. "Mitigating Multi-target Attacks in Hash-Based Signatures." In: *PKC 2016: 19th International Conference on Theory and Practice of Public Key Cryptography, Part I*. Ed. by Chen-Mou Cheng, Kai-Min Chung, Giuseppe Persiano, and Bo-Yin Yang. Vol. 9614. Lecture Notes in Computer Science. Taipei, Taiwan: Springer, Heidelberg, Germany, Mar. 6–9, 2016, pp. 387–416. DOI: 10.1007/978-3-662-49384-7_15. IACR ePrint: ia.cr/2015/1256 (cit. on p. 29).

[22] Panos Kampanakis, Cameron Bytheway, Bas Westerbaan, and Martin Thomson. *Suppressing CA Certificates in TLS 1.3*. Internet-Draft draft-kampanakis-tls-scas-latest-03. Work in Progress. Internet Engineering Task Force, Jan. 2023. 10 pp. URL: https://datatracker.ietf.org/doc/draft-kampanakis-tls-scas-latest/03/ (cit. on pp. 5, 27).

[23] Matthias J. Kannwischer, Peter Schwabe, Douglas Stebila, and Thom Wiggers. "Improving Software Quality in Cryptography Standardization Projects." In: *SSR 2022: Security Standardization Research (2022 IEEE S&P Workshops)*. Genoa, Italy: IEEE Computer Society, June 6–10, 2022, pp. 19–30. DOI: 10.1109/EuroSPW55150.2022.00010. URL: https://wggrs.nl/p/pqclean (cit. on p. 1).

[24] *Kyber*. CRYSTALS-Kyber team. Dec. 23, 2020. URL: https://pq-crystals.org/kyber/ (visited on Mar. 6, 2023), archived at https://web.archive.org/web/20230425044723/https://pq-crystals.org/kyber/ on Apr. 25, 2023 (cit. on p. 14).

[25] Ben Laurie, Adam Langley, and Emilia Kasper. *Certificate Transparency*. RFC 6962. June 2013. DOI: 10.17487/RFC6962 (cit. on p. 27).

[26] Vadim Lyubashevsky, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Peter Schwabe, Gregor Seiler, Damien Stehlé, and Shi Bai. *CRYSTALS-DILITHIUM*. Tech. rep. National Institute of Standards and Technology, 2022. URL: https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022, archived at https://web.archive.org/web/20230429160244/https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022 on Apr. 29, 2023 (cit. on p. 4).

[27] David McGrew, Michael Curcio, and Scott Fluhrer. *Leighton-Micali Hash-Based Signatures*. RFC 8554. Apr. 2019. DOI: 10.17487/RFC8554 (cit. on p. 3).

[28] National Institute of Standards and Technology. *SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions.* Tech. rep. FIPS 202. Gaithersburg, MD, USA: National Institute of Standards and Technology, Aug. 4, 2015. DOI: 10.6028/NIST.FIPS.202 (cit. on p. 29).

[29] National Security Agency. *Announcing the Commercial National Security Algorithm Suite 2.0.* Sept. 7, 2022. URL: https://media.defense.gov/2022/Sep/07/2003071834/-1/-1/0/CSA_CNSA_2.0_ALGORITHMS_.PDF (visited on Mar. 3, 2023), archived at https://web.archive.org/web/20230315013839/https://media.defense.gov/2022/Sep/07/2003071834/-1/-1/0/CSA_CNSA_2.0_ALGORITHMS_.PDF on Mar. 15, 2023 (cit. on p. 21).

[30] Christian Paquin, Douglas Stebila, and Goutam Tamvada. "Benchmarking Post-quantum Cryptography in TLS." In: *Post-Quantum Cryptography - 11th International Conference, PQCrypto 2020.* Ed. by Jintai Ding and Jean-Pierre Tillich. Paris, France: Springer, Heidelberg, Germany, Apr. 15–17, 2020, pp. 72–91. DOI: 10.1007/978-3-030-44223-1_5. IACR ePrint: ia.cr/2019/1447 (cit. on pp. 2, 3, 27).

[31] Thomas Prest, Pierre-Alain Fouque, Jeffrey Hoffstein, Paul Kirchner, Vadim Lyubashevsky, Thomas Pornin, Thomas Ricosset, Gregor Seiler, William Whyte, and Zhenfei Zhang. *FALCON.* Tech. rep. National Institute of Standards and Technology, 2022. URL: https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022, archived at https://web.archive.org/web/20230429160244/https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022 on Apr. 29, 2023 (cit. on p. 4).

[32] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems." In: *Communications of the Association for Computing Machinery* 21.2 (Feb. 1, 1978), pp. 120–126. DOI: 10.1145/359340.359342 (cit. on p. 4).

[33] Jan Rüth, Christian Bormann, and Oliver Hohlfeld. "Large-Scale Scanning of TCP's Initial Window." In: *IMC 2017: 2017 Internet Measurement Conference.* London, United Kingdom: ACM Press, 2017, pp. 304–310. DOI: 10.1145/3131365.3131370. URL: https://conferences.sigcomm.org/imc/2017/papers/imc17-final43.pdf (cit. on p. 27).

[34] Stefan Santesson, Michael Myers, Rich Ankney, Ambarish Malpani, Slava Galperin, and Carlisle Adams. *X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP.* RFC 6960. June 2013. DOI: 10.17487/RFC6960 (cit. on p. 27).

[35] Peter Schwabe, Roberto Avanzi, Joppe Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M. Schanck, Gregor Seiler, Damien Stehlé, and Jintai Ding. *CRYSTALS-KYBER.* Tech. rep. National Institute of Standards and Technology, 2022. URL: https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022, archived at https://web.archive.org/web/20230429160244/https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022 on Apr. 29, 2023 (cit. on p. 4).

[36] Peter Schwabe, Douglas Stebila, and Thom Wiggers. "More Efficient Post-quantum KEMTLS with Pre-distributed Public Keys." In: *ESORICS 2021: 26th European Symposium on Research in Computer Security, Part I.* Ed. by Elisa Bertino, Haya Shulman, and Michael Waidner. Vol. 12972. Lecture Notes in Computer Science. Updated version available via url and IACR ePrint. Darmstadt, Germany: Springer, Heidelberg, Germany, Oct. 4–8, 2021, pp. 3–22. DOI: 10.1007/978-3-030-88418-5_1. IACR ePrint: ia.cr/2021/779. URL: https://wggrs.nl/p/kemtlspdk (cit. on p. 1).

[37] Peter Schwabe, Douglas Stebila, and Thom Wiggers. "Post-Quantum TLS Without Handshake Signatures." In: *ACM CCS 2020: 27th Conference on Computer and Communications Security.* Ed. by Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna. Updated version available via url and IACR ePrint. Virtual Event, USA: ACM Press, Nov. 9–13, 2020, pp. 1461–1480. DOI: 10.1145/3372297.3423350. IACR ePrint: ia.cr/2020/534. URL: https://wggrs.nl/p/kemtls (cit. on p. 1).

[38] Dimitrios Sikeridis, Panos Kampanakis, and Michael Devetsikiotis. "Assessing the Overhead of Post-Quantum Cryptography in TLS 1.3 and SSH." In: *CoNEXT '20: 16th International Conference on Emerging Networking EXperiments and Technologies.* Barcelona, Spain: Association for Computing Machinery, 2020, pp. 149–156. DOI: 10.1145/3386367.3431305. URL: https://www.researchgate.net/publication/346646724_Assessing_the_Overhead_of_Post-Quantum_Cryptography_in_TLS_13_and_SSH (cit. on p. 27).

[39] Dimitrios Sikeridis, Panos Kampanakis, and Michael Devetsikiotis. “Post-Quantum Authentication in TLS 1.3: A Performance Study.” In: *ISOC Network and Distributed System Security Symposium – NDSS 2020*. Updated version available on IACR ePrint. San Diego, CA, USA: The Internet Society, Feb. 23–26, 2020. IACR ePrint: ia.cr/2020/071 (cit. on pp. 27, 29).

[40] Douglas Stebila and Michele Mosca. “Post-quantum Key Exchange for the Internet and the Open Quantum Safe Project.” In: *SAC 2016: 23rd Annual International Workshop on Selected Areas in Cryptography*. Ed. by Roberto Avanzi and Howard M. Heys. Vol. 10532. Lecture Notes in Computer Science. St. John’s, NL, Canada: Springer, Heidelberg, Germany, Aug. 10–12, 2016, pp. 14–37. DOI: 10.1007/978-3-319-69453-5_2. IACR ePrint: ia.cr/2016/1017. URL: https://github.com/open-quantum-safe/ (cit. on p. 27).

[41] Bas Westerbaan. *Sizing up post-quantum signatures*. Post on the Cloudflare blog. Cloudflare, Nov. 2021. URL: https://blog.cloudflare.com/sizing-up-post-quantum-signatures/ (visited on Dec. 22, 2022), archived at https://web.archive.org/web/20230425041856/https://blog.cloudflare.com/sizing-up-post-quantum-signatures/ on Apr. 25, 2023 (cit. on pp. 1, 27).

[42] Wireshark developers. *Wireshark*. URL: https://www.wireshark.org (visited on May 9, 2023) (cit. on p. 3).