The Performance of Post-Quantum TLS 1.3

Markus Sosnowski
Technical University of Munich, Germany

Florian Wiedner
Technical University of Munich, Germany

Eric Hauser
Technical University of Munich, Germany

Lion Steger
Technical University of Munich, Germany

Dimitrios Schoinianakis
Nokia Bell Labs
Athens, Greece

Sebastian Gallenmüller
Technical University of Munich, Germany

Georg Carle
Technical University of Munich, Germany

ABSTRACT

Quantum Computers (QCs) differ radically from traditional computers and can efficiently solve mathematical problems fundamental to our current cryptographic algorithms. Although existing QCs need to accommodate more qubits to break cryptographic algorithms, the concern of "Store-Now-Decrypt-Later" (i.e., adversaries store encrypted data today and decrypt them once powerful QCs become available) highlights the necessity to adopt quantum-safe approaches as soon as possible. In this work, we investigate the performance impact of Post-Quantum Cryptography (PQC) on TLS 1.3. Different signature algorithms and key agreements (as proposed by the National Institute of Standards and Technology (NIST)) are examined through black- and white-box measurements to get precise handshake latencies and computational costs per participating library. We emulated loss, bandwidth, and delay to analyze constrained environments. Our results reveal that HQC and Kyber are on par with our current state-of-the-art, while Dilithium and Falcon are even faster. We observed no performance drawback from using hybrid algorithms; moreover, on higher NIST security levels, PQC outperformed any algorithm in use today. Hence, we conclude that post-quantum TLS is suitable for adoption in today's systems.

CCS CONCEPTS

• **Networks** → **Network performance analysis**; • **Security and privacy** → *Cryptography.*

KEYWORDS

TLS, performance measurements, post-quantum cryptography

ACM Reference Format:
Markus Sosnowski, Florian Wiedner, Eric Hauser, Lion Steger, Dimitrios Schoinianakis, Sebastian Gallenmüller, and Georg Carle. 2023. The Performance of Post-Quantum TLS 1.3. In *Companion of the 19th International Conference on emerging Networking EXperiments and Technologies (CoNEXT Companion '23), December 5–8, 2023, Paris, France.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3624354.3630585

1 INTRODUCTION

Quantum Computers (QCs) are no longer perceived as a conjecture of computational sciences and theoretical physics. Considerable research efforts and enormous corporate and government funding are invested in practical QCs. Examples of accelerated efforts toward large-scale QCs include Google's announcement of achieving quantum supremacy [1] and IBM's latest 433-qubit processor [9]. The existence of a QC would mark a cornerstone in technological evolution. It would mean that some computational problems considered intractable for today's conventional computers become tractable for QCs. Unfortunately, two of the few practical algorithms that can be efficiently solved on QCs are the mathematical problems constituting the basis of today's public-key cryptography, namely the integer factorization and the discrete logarithm problem [23]. Thus, most of today's public-key algorithms, including Rivest–Shamir–Adleman (RSA), Diffie-Hellman (DH), and ECDH, as well as the accompanying digital Signature Algorithms (SAs), would need to be replaced by ones that offer cryptanalytic resistance against QCs. To mitigate this threat, the National Institute of Standards and Technology (NIST) launched a competition in 2016 to identify and standardize novel algorithms based on mathematical problems resistant to QCs, commonly referred to as Post-Quantum Cryptography (PQC) [22]. At the end of 2022 and after three evaluation rounds, winners (Kyber, Dilithium, Falcon, and SPHINCS+) were announced [16]. Their standardization is expected by 2024. NIST also announced a fourth round with three Key Agreements (KAs) still under consideration: HQC, Bike, and Classic McEliece.

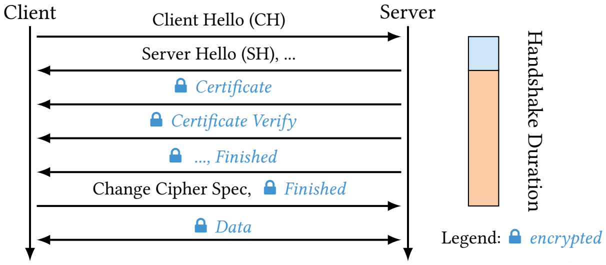


Figure 1: TLS 1.3 handshake, including the two phases where latency can be measured without decryption.

The problem of "Store-Now-Decrypt-Later" (adversaries storing encrypted data today, may be able to decrypt them once powerful QCs become available) essentially dictates that every day we lose today by not deploying a quantum-safe solution might correspond to exposed data in the future. As such, PQC becomes increasingly relevant, and selecting the best-performing algorithms for actual applications like Transport Layer Security (TLS) gets important.

This work investigates the performance of pre- and post-quantum algorithms in TLS, providing the following contributions: *i)* a comparison of traditional and Post-Quantum (PQ) algorithms used for

CoNEXT Companion '23, December 5–8, 2023, Paris, France

ACM ISBN 979-8-4007-0407-9/23/12…$15.00
https://doi.org/10.1145/3624354.3630585

the KA and as SAs in TLS using the same measurement methodology on identical hardware for comparability; *ii)* black- and white-box measurements providing end-to-end handshake latency, transmitted data volumes, and insights into causes of observed behavior revealing potential bottlenecks; *iii)* findings on how the choice of the seemingly independent KA and SA can influence each other; *iv)* recommendations on the best-suited algorithms from a performance perspective; and *v)* published experiment scripts and raw measurement data to enable reproducible results and to support the community (*cf.*, Appendix B).

2 BACKGROUND

This work focuses on TLS 1.3, the most popular option for establishing secure connections between two endpoints [32]. TLS uses algorithms affected by PQ for the initial KA, the handshake signature, and the signatures forming the x509 Public Key Infrastructure (PKI). After a completed handshake, TLS switches to symmetric encryption, which is only partly affected by PQ cryptanalysis. Grover developed an algorithm which, theoretically, halves the security level of symmetric encryption [13]. However, the implications for symmetric algorithms are out of this paper's scope.

Transport Layer Security (TLS). TLS 1.3 is independent of concrete KAs or SAs. For TLS, they are just negotiated parameters of the handshake. Hence, PQ-safe TLS only differs from traditional TLS by announcing and selecting PQC during the handshake. Clients can pre-compute a key-share for the KA it expects servers to select, enabling 1-Round Trip Time (RTT) handshakes. This behavior is typical for TLS 1.3, reducing unnecessary delays while providing perfect forward secrecy. Therefore, we focus on 1-RTT handshakes and configured TLS such that the 2-RTT fallback never occurred. The server certificate determines the SA, and we used specific certificates to investigate a certain SA. Figure 1 describes a 1-RTT TLS 1.3 handshake, lists the TLS messages important for this work, and marks the investigated handshake duration. Clients initiate handshakes with the Client Hello (CH), and servers respond with the Server Hello (SH). Both messages perform the KA and are, therefore, affected by PQ. After the KA is finished, the rest of the handshake is performed over a symmetrically encrypted channel. The Certificate message is necessary for the client to authenticate the server. The included certificates can use PQ signatures; hence, their size is affected by PQ. Subsequently, the Certificate Verify message transmits the handshake signature performed with the server certificate's private key. Again, this signature is affected by PQ. At last, the server concludes the handshake with a Handshake Finished (HF) message, including a hash over all messages to protect the integrity of the handshake. Depending on the Maximum Transmission Unit (MTU), the messages from the SH up to the HF message can be sent in a single IP packet and it is open to the implementation how to combine them efficiently. With the help of the received certificates, the client verifies the signature and whether the server is trustworthy according to the PKI. To complete the handshake, the client sends a dummy Change Cipher Spec and an HF message; again, containing a hash over previous messages. Both were always combined in the same IP packet in our measurements. A passive observer can measure the first (CH to SH) and the second part (SH to Client Finished) of the handshake because they contain unencrypted data.

Table 1: Security levels and requirements [17]

Level	Comparable difficulty to a …
1:	key search on a block cipher with a 128-bit key (e.g., AES128)
2:	collision search on a 256-bit hash function (e.g., SHA256)
3:	key search on a block cipher with a 192-bit key (e.g., AES192)
4:	collision search on a 384-bit hash function (e.g., SHA384)
5:	key search on a block cipher with a 256-bit key (e.g., AES256)

Post-Quantum Cryptography (PQC). PQC is based on mathematical problems considered intractable for QCs. The NIST has started a challenge across multiple rounds for researchers to submit candidates or attacks for PQC. Recently, they announced to standardize four PQ finalists (Kyber, Dilithium, Falcon, and SPHINCS+) and promoted four additional algorithms (Bike, Classic McEliece, HQC, and Sike) to a fourth round [16]. However, Sike was broken shortly after the announcement [7]. Additionally, Classic McEliece uses key sizes exceeding the $2^{16} - 1$ B anticipated in TLS 1.3 [21] and would require TLS protocol changes. Therefore, we excluded both KAs from our list of investigated PQC. All algorithms can be configured to fulfill different security or performance requirements. The respective authors provide several pre-selected configurations that can be directly used in a TLS handshake depending on the required security level. According to NIST [17], an algorithm fulfills a certain level if its computational complexity is comparable to specific problems (*cf.*, Table 1). The security of cryptographic algorithms is hard to prove. Usually, they are "proven over time," *i.e.*, they are considered secure because no vulnerabilities are known. The longer researchers tried to find such vulnerabilities, the higher the chances that none exist. This causes two challenges for the adoption of the comparatively new PQC: *(i)* unknown vulnerabilities might exist, threatening the security of encrypted communication and *(ii)* even if none exist, it is hard to convince people that the new algorithms are secure and should be adopted. A hybrid approach mitigates this problem by combining two KAs in a way that both must be broken, before an attacker can reclaim the shared secret [29]. The two KAs are performed independently and the final shared secret is a concatenated version of the two individual secrets. Similarly, two SAs can be combined [18] such that both need to be broken to threaten the security of the PKI. We call the resulting combination using either approach a hybrid. They have the advantage that they are, per design, as secure as the current state-of-the-art; moreover, they benefit from the added security of PQC. As a downside, the overhead of combining algorithms can increase the costs of TLS.

3 RELATED WORK

The research interest in PQ TLS is considerable and publications, especially on performance, are numerous.

In 2020, Paquin et al. [19] evaluated different experimental setups with emulated and real-world network conditions. They found that handshake completion times mainly depend on the speed of cryptographic operations under good and data size under bad network conditions. In the same year, Sikeridis et al. [25] published their work on PQ signatures and authentication. They evaluated cryptographic performance with Internet-wide measurements and showed that both key and certificate size impact the handshake time significantly. Sikeridis et al. [24] further investigated the performance impact of PQC on TLS and SSH handshakes. They measured

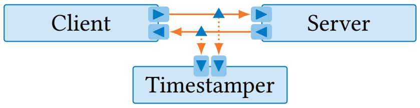


Figure 2: Measurement setup

an increased delay that can be mitigated by increasing the TCP initial window size to achieve competitive performance to traditional algorithms. Paul et al. [20] investigated mixing algorithms in an x509 PKI. They showed promising results with a combination of XMSS for root certificates and Dilithium for the rest of the certificate chain. Focusing on low-power embedded systems, Bürstinghaus-Steinbach et al. [6] evaluated Kyber and SPHINCS+ against their traditional counterparts, ECDH and ECDSA. Because some algorithms, especially Kyber, outperformed non-PQ variants, they argued deploying PQCs is feasible with little overhead. Continuing the focus on embedded devices, Marchsreiter and Sepúlveda [14] analyzed hybrid PQ algorithms and found that while PQC can be more efficient, handshake times can still be negatively impacted by choosing hybrid algorithms. They argued that the performance on constrained devices might suffer from large key sizes and a slower connection. In 2022, Tzinos et al. [31] evaluated a variety of KAs in a local network setting. They concluded that their results align with the choices of NIST, suggesting Kyber as a new standard. Measurements by Cloudflare [33] showed the competitive performance of PQ algorithms. Further, Cloudflare offers hybrid KAs (currently, only Kyber [8] and no SAs) as a beta service to its customers. Recently, claims [5] were made that the Kyber security calculation is flawed; thus, reducing Kyber-512 below NIST level 1. Therefore, Kyber may require longer keys and longer calculation times to actually achieve comparable security.

4 MEASUREMENT METHODOLOGY

We measured the performance of PQCs in TLS through a series of sequential TLS handshakes in 60 s intervals. In TLS 1.3, the KA and SA are selected independently. However, a TLS handshake requires both, demanding a combined measurement and a specific combination of KA and SA. We analyzed the four general KAs: ECDH, HQC, Kyber, and Bike, each available in multiple variants, resulting in 23 KAs. Similarly, the four general SAs (RSA, Falcon, Dilithium, and SPHINCS+) result in 22 investigated SAs. SPHINCS+ offers 36 variants causing latencies between a few milliseconds and several seconds. Our paper considers only the fastest SPHINCS+ configuration (simple haraka signature optimized for signing speed) and subsets of the theoretical $|\mathrm{KA} \times \mathrm{SA}|$ combinations. Further SPHINCS+ and KA-SA combinations can be found according to Appendix B. Each measurement was performed twice: as *i)* black-box and *ii)* white-box measurements. The former ran without interference of other utilities, while the latter analyzed CPU usage.

Measurement setup. Figure 2 shows our measurement setup. Client and server are separate hosts, directly connected via 10 Gbit/s fiber links. The third host (timestamper) tapped into the connection via passive optical fiber taps. This three-node setup allowed the collection of precise hardware timestamps for all packets exchanged between client and server with minimal impact on latency and jitter due to passive tapping. All nodes used identical hardware, Intel Xeon D-1518 SoCs (4 cores, 2.2 GHz), dual-port Intel X552 NICs, and 32 GB of RAM. Client and server ran Debian Bullseye (Linux kernel v5.10). For TLS offering support for PQC, we used a fork of OpenSSL [3]. We measured the TLS handshakes using the integrated OpenSSL client and server. The timestamper ran Debian buster (Linux kernel v4.19) and MoonGen [12] to record hardware timestamps. Our measurements and analyses were automated, ensuring the repeatability of our results. We measured the handshake performance for 60 s. During the measurement period, we observed between 1 k and 30 k handshakes, depending on the complexity of the investigated algorithms. The reported latencies are the median values taken during each measurement period.

Black-Box Measurements. Only the TLS server and client were running on the respective nodes, the timestamper node monitored the connection. As described in Figure 1, we measured two parts of the entire handshake. Because our TLS library always sent the "Finished" message with the unencrypted "Change Cipher Spec" message, we could accurately timestamp the three events without decrypting the traffic. The first part includes computations related to the KA on the server. We observed situations where the server implementation withheld the SH until it computed the handshake signature. In these cases, the first part additionally depends on the SA. We analyzed such cases in Section 5.2. The second part depends both on the KA and SA because of the necessary key computation and signature verification on the client. Left out were any CH-related computations on the client.

White-Box Measurements. We used the profiling tool Linux perf to analyze delays and computational costs. Linux perf halts code execution in a predefined frequency and traces the current call stack. Over time, perf recordings allow reconstructing the execution times of individual functions. We pinned the client/server applications to dedicated cores, avoiding unnecessary jitter. However, collecting stack traces can impact application performance. Therefore, any latency recorded by perf may differ from the black-box measurements. After recording, Linux perf offers various possibilities to sort, combine, and filter the stack traces, *e.g.*, collapse and group them according to the shared object or library.

Optimized TLS Message Buffering In our study, we identified inconsistent behavior of our OpenSSL library: for specific algorithms, the server occasionally buffered all computed TLS messages and sent them to the client as a batch (depending on the MTU in a single IP packet); or, sent some messages earlier. This behavior was caused by an internal library buffer of 4096 B. By default, the library would compute all messages and forward them to the client only after the Certificate Verify message was completed; however, whenever the buffer size was exceeded by a message, its content (notably, the SH) was flushed to the TCP stack and forwarded to the client. This behavior impacts the handshake latency; especially for expensive key decapsulations on the client. Overall performance improves if clients start computing the secret key while servers still compute the rest of the handshake. To improve consistency, we modified the OpenSSL library to immediately send the SH and the Certificate Message to the client as soon as they are computed.

5 EVALUATION

The following sections present our measurement results. It is only possible to measure the TLS handshake performance impacted by both KAs and SAs because one cannot work without the other. To evaluate them individually, we fix KA and SA to X25519 and RSA

Table 2: Handshakes latency, data usage, and total number conducted in the one minute period per NIST security level

(a) KAs combined with rsa:2048 as SA

Lvl	KA	Handshake		Data Sent (B)	
		Latency Median (ms)	# Total	Client	Server
1	**X25519**	0.25, 1.48	22.3k	689	1455
	bikel1	0.24, 2.79	13.4k	2250	3048
	hqc128	0.27, 1.48	21.9k	2958	6060
	kyber512	0.20, 1.78	20.8k	1457	2191
	kyber90s512	0.19, 1.79	20.5k	1457	2191
	p256	0.33, 1.50	21.2k	722	1488
	p256_bikel1	0.42, 2.58	13.5k	2315	3113
	p256_hqc128	0.52, 1.31	20.5k	3023	6125
	p256_kyber512	0.51, 1.81	17.9k	1522	2256
3	bikel3	0.42, 6.07	6.9k	3844	4642
	hqc192	0.53, 1.40	19.5k	5387	10 761
	kyber768	0.25, 1.82	19.2k	1893	2511
	kyber90s768	0.20, 1.78	20.7k	1893	2511
	p384	3.09 2.63	7.0k	754	1520
	p384_bikel3	3.23 9.00	3.7k	3941	4739
	p384_hqc192	3.39 3.30	6.1k	5484	10 858
	p384_kyber768	3.17 2.72	6.7k	1990	2608
5	hqc256	0.72, 1.92	15.3k	8214	16 412
	kyber1024	0.25, 1.78	19.8k	2277	3043
	kyber90s1024	0.22, 1.78	20.0k	2277	3043
	p521	6.97 5.30	3.4k	790	1556
	p521_hqc256	7.52 9.38	2.4k	8347	16 545
	p521_kyber1024	7.06 5.41	3.3k	2410	3176

(b) SAs combined with X25519 as KA

Lvl	SA	Handshake		Data Sent (B)	
		Latency Median (ms)	# Total	Client	Server
	rsa:1024	0.32, 0.68	30.0k	689	1066
	rsa:2048	0.25, 1.48	22.3k	689	1455
1	falcon512	0.36, 1.02	24.3k	689	2920
	rsa:3072	0.26, 3.41	12.6k	689	1839
	rsa:4096	0.25, 6.88	7.4k	689	2223
	sphincs128	0.28 15.02	3.7k	1001	36 153
	p256_falcon512	0.39, 1.35	21.7k	689	3137
	p256_sphincs128	0.28 15.48	3.5k	1001	36 372
2	dilithium2	0.39, 0.84	27.1k	689	6981
	dilithium2_aes	0.39, 0.78	27.5k	689	6981
	p256_dilithium2	0.39, 1.27	22.6k	689	7185
3	dilithium3	0.36, 0.94	26.1k	741	9471
	dilithium3_aes	0.38, 0.86	27.0k	741	9471
	sphincs192	0.27 23.83	2.4k	1105	74 769
	p384_dilithium3	0.31, 3.86	11.6k	741	9823
	p384_sphincs192	0.27 28.75	2.0k	1105	75 087
5	dilithium5	0.36, 1.10	24.1k	793	12 923
	dilithium5_aes	0.36, 0.98	25.2k	793	12 923
	falcon1024	0.38, 1.89	18.4k	689	5097
	sphincs256	0.27 49.52	1.2k	1209	104 253
	p521_dilithium5	0.29, 7.55	6.8k	793	13 330
	p521_falcon1024	0.35, 8.72	6.0k	689	5572
	p521_sphincs256	0.27 60.78	1.0k	1209	104 679

Legend: **Pre-Quantum** Hybrid □ CH → SH ▨ SH → Client Finished

2048 for most analyses, respectively. Unless stated otherwise, we used the optimized version of OpenSSL, described in Section 4.

5.1 Black-box Measurements

For this analysis, we measured the latency and the amount of data exchanged between client and server for a TLS 1.3 handshake. The black-box approach does not deploy any analysis tools, resulting in unbiased measurements. Results for the investigated KAs and SAs are listed in Tables 2a and 2b. Algorithms with an underscore (_) are hybrids. The depicted latency shows the median time to complete the handshake split into two phases (cf. Figure 1). Lastly, we present the data volume client or server sent in each handshake.

Results showed that PQC (except Bike and SPHINCS+) challenge traditional algorithms on security level one, while greatly outperforming them on higher levels. Moreover, we observed almost no overhead in using hybrid algorithms. On level one, the hybrids are effectively as fast as Elliptic Curve Cryptography (ECC) or PQC. The pre-quantum algorithms (except Bike) bottleneck the hybrids on higher levels. Handshakes with Dilithium, regardless of the security level, were faster than our current state-of-the-art rsa:2048 that is considered sub-level one (effectively correlating to a 112-bit rather than the required 128-bit key length [4]). We observed minor performance improvements using variants of Kyber and Dilithium (indicated as 90s or aes), which use AES instead of SHAKE for masking and sampling. Even the fastest SPHINCS+ variant used in Table 2b offered low performance; handshake latency and data usage (due to large signatures) were up to 20 times higher.

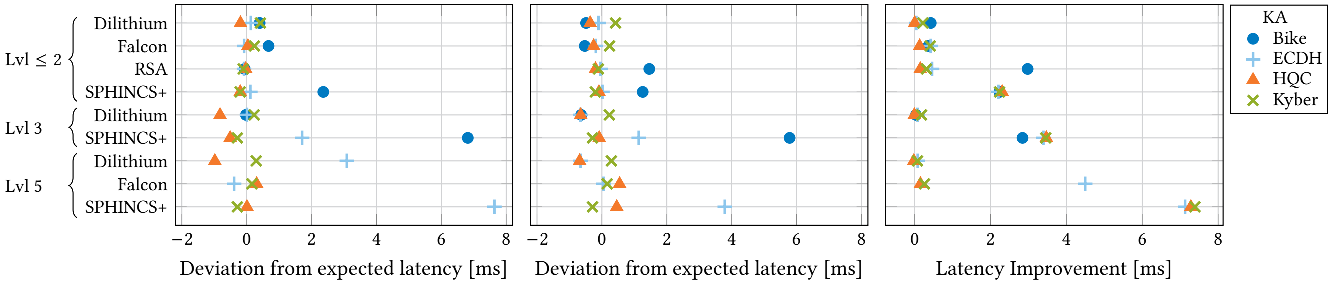
5.2 Independence of KA / SA

KAs and SAs are independently selected in TLS 1.3 and we assumed that they influence the handshake independently. However, this was not always the case based on the following experiment.

If we assume that the KA and SA independently impact the TLS handshake and the latency caused by each algorithm is deterministic, we should be able to predict the handshake latency of arbitrary combinations. Otherwise, they somehow influenced each other. Given two KAs $k1, k2$ and two SAs $s1, s2$, we can measure the TLS handshake latency $M : KA \times SA \rightarrow \mathbb{R}$. If both are independent, then $M(k1, s1) + M(k2, s2) = M(k1, s2) + M(k2, s1)$. Hence, we can calculate an expected value for arbitrary combinations with our baseline $E : KA \times SA \rightarrow \mathbb{R}$ with $E(k, s) = M(k, \text{rsa}_{2048}) + M(\text{X25519}, s) - M(\text{X25519}, \text{rsa}_{2048})$. We combined all KAs and SAs (except hybrids and only rsa:3072) on their respective NIST security level and calculated the deviation $E(k, s) - M(k, s)$ for both default and optimized OpenSSL version—which sends SHs immediately after computation—in Figures 3a and 3b, respectively.

While we expected small deviations, the results reveal that specific combinations significantly impacted each other. Dilithium-HQC combinations were slightly slower than expected, Bike and ECDHs were faster, depending on the SA. Though we cannot rule out additional factors, we found this was due to the TLS message buffering of OpenSSL (*cf.*, Section 4). It introduces a dependency between KA-SA and causes deviations from the calculated expected latency. We found two explanations: first, algorithm combinations were slightly faster if their combined key sizes caused an early push of the SH, although, one algorithm alone would not trigger the push. The outliers for the optimized OpenSSL version are smaller because this push was consistent (see Figure 3b). Second, CPU-intensive KAs (*i.e.*, Bike, or ECDH above level one) benefited from the parallel processing enabled by the early SH; especially, when the SA is computationally heavy as well (*e.g.*, for SPHINCS+ and RSA 3072). This is the dominating factor for both OpenSSL versions. In the case of Bike and RSA, the effect is only visible for the optimized version because they did not cause a push in the default version.

Aligning the OpenSSL behavior had an interesting effect: most handshakes were faster, as shown in Figure 3c. Combinations with SPHINCS+ improved up to 7.4 ms; indicating optimization potential libraries could utilize for some algorithms. We showed the handshake latency can be reduced when the SH is pushed to the client



(a) Default OpenSSL behavior **(b) Optimized OpenSSL behavior** **(c) Improvement of the optimized behavior**

Figure 3: Comparison of the handshake latency for KA-SA combinations on the same level. The OpenSSL behavior of assembling TLS messages (default and optimized) influences the latency. A positive deviation means it was faster-than-predicted.

Table 3: Results from the white-box measurements

Lvl	KA	SA	HS [1/s]	CPU Cost [ms]		Packets Sent		Server Library Distribution	Client Library Distribution
				Server	Client	Server	Client		
≤ 2	**`X25519`**	**`rsa:2048`**	402	3.1	1.9	3	1	77.37 / 13.27	45.54 / 34.62 / 8.7 / 7.5
	`kyber512`	`dilithium2`	473	1.6	1.7	6	1	49.71 / 28.16 / 11.74 / 6.4	40.63 / 34.01 / 14.16 / 8.7
	`bikel1`	`dilithium2`	231	1.8	6.5	7	2	48.17 / 30.51 / 13.27	74.21 / 11.10 / 9.35
	`kyber512`	`sphincs128`	59	24.1	5.1	27	1	95.62	77.24 / 14.53
	`hqc128`	`falcon512`	370	2.3	2.5	7	2	59.29 / 19.49 / 13.10 / 6.2	35.98 / 28.80 / 25.79 / 6.5
	`p256_kyber512`	`dilithium2`	382	2.0	2.2	6	1	54.87 / 23.38 / 10.83 / 7.1	48.25 / 29.02 / 10.78 / 9.48
3	`kyber768`	`dilithium3`	403	1.8	1.9	8	2	53.26 / 26.52 / 11.38 / 6.3	44.49 / 30.14 / 13.67 / 8.2
5	`kyber1024`	`dilithium5`	419	2.0	2.3	13	2	54.20 / 25.27 / 12.09	48.83 / 28.65 / 13.28 / 7.1

Legend: **Pre-Quantum** Hybrid libcrypto.so.1.1 kernel.kallsyms libssl.so.1.1 libc-2.31.so ixgbe python3.9

as soon as it is computed. This introduces a dependency between KA-SA; however, it is low compared to the whole handshake.

5.3 White-box Measurements

The white-box measurement results are displayed in Table 3, showing a small selection of analyzed KA and SA pairs. To access the entire table, refer to Appendix B. In addition to the profiling results in the columns *CPU cost*, *server-*, and *client library distribution*, we added the handshake rate and the number of transmitted packets for better comparison. The CPU cost is derived from the total CPU time divided by the total number of handshakes. The library distributions show the time spent in each library during the experiment. The libraries with the largest shares are: *i)* `libcrypto`: Summing up the computing time spent on cryptographic functions; *ii)* `kernel`: Computing time spent in the kernel, e.g., for packet processing; *iii)* `libssl`: Comprising the computing time spent on TLS protocol functions; *iv)* `libc`: C standard library functions; *v)* `ixgbe`: Processing in the NIC driver; *vi)* `python`: Programs used for the testbed tools performing the experiment. At first glance, libcrypto, kernel, and libssl typically dominate with an approx. portion of 90 %. Ixgbe, python, and libc have a minor impact on the overall handshake performance. We used the PQ combination of X25519 and RSA as baseline. For it, we observed that the server-side computations are more resource-intensive than the client-side. Based on the perf profiling, we identify the computations in libcrypto as root cause. The combination of the PQC Kyber and Dilithium performed well, and we observed only minimal performance decreases on higher levels. The combination of Dilithium and Bike performed good on the server-side but lacked performance on the client-side. In this case, the libssl dominated the computation on the client. Rather than using the cryptographic functions of libcrypto, the Bike client is predominantly implemented in libssl. Especially for low-power embedded systems, increased CPU costs are problematic. In contrast, the combination of Kyber and SPHINCS+ required significantly more resources on the server. In this case, libcrypto dominated with over 95 % of the total computational time. Still, the client had overall CPU cost of 5.1 ms, second-worst performance in this comparison. HQC in combination with Falcon, showed reasonable performance on both ends even though the client spent more time in libssl. Furthermore, the hybrid algorithm of Kyber, in combination with Dilithium, showed a good performance.

All in all, the presented white-box measurements reveal whether the server or the client required more resources. Moreover, the library distribution exposes the CPU costs each library consumed.

5.4 Constrained Environments

Up to this point, we have used a loss-free, low-latency connection with high bandwidth (*cf.*, Section 4). This allowed observing TLS and PQC performance under beneficial conditions. To investigate realistic network conditions, we extended our analyses to constrained environments considering loss, delay, and limited bandwidths; *e.g.*, faced by embedded devices or wireless communication.

We used netem [30] to emulate different scenarios: dealing with high loss (10%), long delay (1 s RTT), low bandwidth (1 Mbit/s), and two scenarios modeled after real-world settings: LTE-M over 15km and a 5G setup. Details are presented in Appendix A, Table 4. We noticed several findings: *i)* a high loss slowed down handshakes, but compared to the other parameters it had the least impact; *ii)* a low bandwidth slowed down all handshakes and had an increased impact on the algorithms that transferred a lot of data (*e.g.*, HQC, Dilithium, or SPHINCS+); *iii)* the latency of TLS grows approximately linearly with added delay; and *iv)* the two realistic scenarios mostly depended on the RTT, although the high loss of the LTE-M scenario caused the median latency to occasional increases by

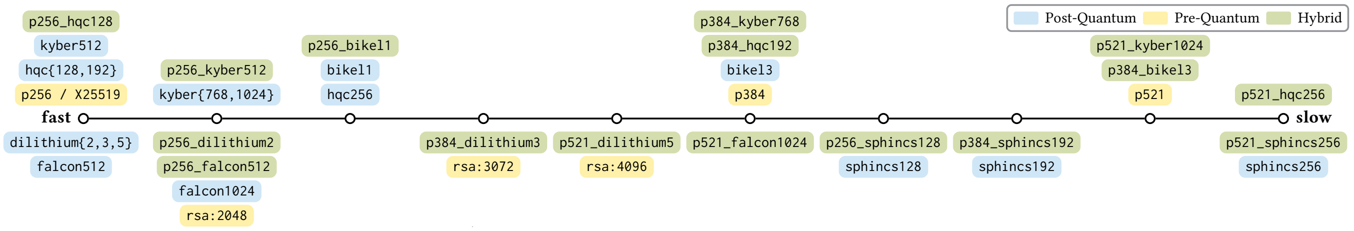


Figure 4: Key Agreements (top) and Signature Algorithms (bottom) ranked depending on the logarithmic handshake latency we measured. Top and Bottom is ranked separately with the algorithms on the left being the fastest.

several RTTs. Note that we measured consecutive handshakes in a 60 s period, which resulted in significantly fewer samples for high-delay scenarios. The 1 s RTT scenario revealed an interesting finding: All SPHINCS+ and HQC and Dilithium on level 5 required multiple RTTs to complete the handshake. This was caused by the TLS messages from the server exceeding the TCP Congestion Window (CWND). Every TLS handshake was conducted after the TCP handshake; thus, the CWND was still at the configured minimum of the Slow Start Phase (usually $10 \times MSS$). Table 2 shows that PQC has an increased data volume, resulting in occasional two to four RTTs for the TLS handshakes. To conclude, the larger PQC keys significantly impact environments with reduced bandwidth or high RTTs. Kyber and Falcon surpass the other PQ algorithms in low-bandwidth settings (*e.g.*, LTE-M) due to shorter keys. We expect the initial CWND will become an important tuning factor for TLS servers to retain the ability for 1-RTT handshakes; especially, when combining PQ KAs and SAs, both increasing handshake size.

5.5 PQ TLS for Attack Scenarios

Our results present PQ TLS as a protocol with a potentially high degree of asymmetry. This might be exploited, *e.g.*, to overload servers utilizing skewed computation costs between client and server [10]. We demonstrated the CPU costs can be up to 6× higher on the server (*cf.*, Table 3). Significant size differences of client requests and server replies may be used to overwhelm targets utilizing spoofed requests [15]. Table 2b lists server replies up to 96× larger than the initial client requests. For comparison, QUIC mandates a maximum amplification factor of 3 to prevent such attacks [15]. The main lever in both attack scenarios was the choice of SA.

6 DISCUSSION OF RESULTS

While we do not evaluate the algorithms' security, we provide precise measurements demonstrating their end-to-end performance. However, our tables are complex and we wanted to refine, simplify, and put our results into context so they can be used as recommendations. Therefore, Figure 4 provides a PQC ranking. We took the overall latency from Tables 2a and 2b, computed the *log*, linearly scaled the results to [0, 10], and rounded them to whole numbers. This results in a ranking of the algorithms with the top ranks on the left. We see that the PQ algorithms challenge our state of the art and some SA are even faster. Moreover, the hybrid algorithms provided no performance downside on NIST level one. On higher levels, the PQ algorithms become faster than the hybrids (which have the pre-quantum algorithms as bottleneck). Note that latency should not always be the main criterion; especially on constrained devices, the clients' computational might be more important. Moreover, in case of low-bandwidth or high-delay environments, the data volume can be more important and Section 5.4 showed that in such cases Kyber and Falcon surpassed other PQ algorithms because of their smaller keys. See Appendix B for additional rankings.

In conclusion, we recommend shifting towards hybrid algorithms as they *i)* provide the security of our current algorithms proven-over-time, *ii)* protect against actors capturing network traffic to extract sensitive data when powerful QCs are available, and *iii)* cause no significant performance drawback. The performance disadvantage of SPHINCS+ is apparent, a "clear drawback" [2], according to its authors. However, its underlying mathematical properties are well understood in the cryptography community. Hence, SPHINCS+ may be considered more secure from today's perspective.

7 CONCLUSION

This work compares the performance of traditional and PQ algorithms for TLS. All measurements were conducted using the same measurement methodology on identical hardware to ensure the comparability of results. The focus of our measurement campaign was the initial handshake of TLS, as it is based on asymmetric cryptography, which is considered broken by QCs in the future. Our results allow to compare the performance of all pre- and post-quantum algorithms currently relevant for TLS. Additionally, we found that PQ handshakes can exceed the initial TCP Congestion Window on servers, resulting in additional round trips per handshake. The initial Congestion Window could become an important tuning factor for PQ TLS. Our findings showed that PQC can be very fast (HQC and Kyber) and challenges the state-of-the-art; regarding authentication, Dilithium and Falcon are even faster than RSA while claiming higher NIST security levels. In low-bandwidth environments, PQC can be slower than the current state-of-the-art due to the increased data volume. On NIST security levels three to five, PQC outperforms all algorithms in use today. In conclusion, there is no significant performance drawback in using PQC or hybrid algorithms. We expect our results will be useful for optimizing TLS libraries and for choosing the right algorithm for each application. We hope the further PQC development considers our performance perspective such that PQ algorithms remain practical for TLS. Ideally, they are a performance improvement of our state-of-the-art, providing an additional incentive for their adoption.

ACKNOWLEDGMENTS

We received funding from European Union's Horizon 2020 research and innovation program (grant No. 101008468 and 101079774), the Bavarian Ministry of Economic Affairs, Regional Development and Energy as part of the project 6G Future Lab Bavaria, and by the German Federal Ministry of Education and Research (BMBF) under the projects 6G-life (16KISK002) and 6G-ANNA (16KISK107).

REFERENCES

[1] Frank Arute, Kunal Arya, Ryan Babbush, Dave Bacon, Joseph Bardin, Rami Barends, Rupak Biswas, Sergio Boixo, Fernando Brandao, David Buell, Brian Burkett, Yu Chen, Jimmy Chen, Ben Chiaro, Roberto Collins, William Courtney, Andrew Dunsworth, Edward Farhi, Brooks Foxen, Austin Fowler, Craig Michael Gidney, Marissa Giustina, Rob Graff, Keith Guerin, Steve Habegger, Matthew Harrigan, Michael Hartmann, Alan Ho, Markus Rudolf Hoffmann, Trent Huang, Travis Humble, Sergei Isakov, Evan Jeffrey, Zhang Jiang, Dvir Kafri, Kostyantyn Kechedzhi, Julian Kelly, Paul Klimov, Sergey Knysh, Alexander Korotkov, Fedor Kostritsa, Dave Landhuis, Mike Lindmark, Erik Lucero, Dmitry Lyakh, Salvatore Mandrà, Jarrod Ryan McClean, Matthew McEwen, Anthony Megrant, Xiao Mi, Kristel Michielsen, Masoud Mohseni, Josh Mutus, Ofer Naaman, Matthew Neeley, Charles Neill, Murphy Yuezhen Niu, Eric Ostby, Andre Petukhov, John Platt, Chris Quintana, Eleanor G. Rieffel, Pedram Roushan, Nicholas Rubin, Daniel Sank, Kevin J. Satzinger, Vadim Smelyanskiy, Kevin Jeffery Sung, Matt Trevithick, Amit Vainsencher, Benjamin Villalonga, Ted White, Z. Jamie Yao, Ping Yeh, Adam Zalcman, Hartmut Neven, and John Martinis. 2019. Quantum Supremacy using a Programmable Superconducting Processor. *Nature* 574 (2019). https://doi.org/10.1038/s41586-019-1666-5

[2] Jean-Philippe Aumasson, Daniel J. Bernstein, Ward Beullens, Christoph Dobraunig, Maria Eichlseder, Scott Fluhrer, Stefan-Lukas Gazdag, Andreas Hülsing, Panos Kampanakis, Stefan Kölbl, Tanja Lange, Martin M. Lauridsen, Florian Mendel, Ruben Niederhagen, Christian Rechberger, Joost Rijneveld, Peter Schwabe, and Bas Westerbaan. 2020. SPHINCS+ Submission to the 2nd round of the NIST post-quantum project. Specification document (part of the submission package). (Oct. 2020).

[3] Michael Baentsch et al. 2023. *OQS-OpenSSL_1_1_1 Repository.* https://github.com/open-quantum-safe/openssl Last accessed: June 26, 2023.

[4] Elaine Barker. 2020. *Recommendation for Key Management Part 1 - General.* National Institute of Standards and Technology. https://doi.org/10.6028/NIST.SP.800-57pt1r5

[5] Daniel J. Bernstein. 2023. *The inability to count correctly: Debunking NIST's calculation of the Kyber-512 security level.* https://blog.cr.yp.to/20231003-countcorrectly.html Last accessed: October 6, 2023.

[6] Kevin Bürstinghaus-Steinbach, Christoph Krauss, Ruben Niederhagen, and Michael Schneider. 2020. Post-Quantum TLS on Embedded Systems: Integrating and Evaluating Kyber and SPHINCS+ with Mbed TLS. In *Proceedings of the ACM Asia Conference on Computer and Communications Security.* https://doi.org/10.1145/3320269.3384725

[7] Wouter Castryck and Thomas Decru. 2022. An efficient key recovery attack on SIDH (preliminary version). *Cryptology ePrint Archive* (2022). https://eprint.iacr.org/2022/975.pdf

[8] Cloudflare Research. 2023. *Cloudflare Research: Post-Quantum Key Agreement.* https://pq.cloudflareresearch.com/ Last accessed: June 27, 2023.

[9] Hugh Collins and Chris Nay. 2023. *IBM Unveils 400 Qubit-Plus Quantum Processor and Next-Generation IBM Quantum System Two.* https://newsroom.ibm.com/2022-11-09-IBM-Unveils-400-Qubit-Plus-Quantum-Processor-and-Next-Generation-IBM-Quantum-System-Two Last accessed: June 26, 2023.

[10] Scott A. Crosby and Dan S. Wallach. 2003. Denial of Service via Algorithmic Complexity Attacks. In *Proceedings of the USENIX Security Symposium.* USENIX Association. https://www.usenix.org/conference/12th-usenix-security-symposium/denial-service-algorithmic-complexity-attacks

[11] Samir Dawaliby, Abbas Bradai, and Yannis Pousset. 2016. In depth performance evaluation of LTE-M for M2M communications. In *Proceedings of the IEEE International Conference on Wireless and Mobile Computing, Networking And Communications (WiMob).* https://doi.org/10.1109/WiMOB.2016.7763264

[12] Paul Emmerich, Sebastian Gallenmüller, Daniel Raumer, Florian Wohlfart, and Georg Carle. 2015. MoonGen: A Scriptable High-Speed Packet Generator. In *Proceedings of the Internet Measurement Conference.* https://doi.org/10.1145/2815675.2815692

[13] Lov K. Grover. 1996. A Fast Quantum Mechanical Algorithm for Database Search. In *Proceedings of the ACM Symposium on Theory of Computing.* https://doi.org/10.1145/237814.237866

[14] Dominik Marchsreiter and Johanna Sepúlveda. 2022. Hybrid Post-Quantum Enhanced TLS 1.3 on Embedded Devices. In *Proceedings of the Euromicro Conference on Digital System Design (DSD).* https://doi.org/10.1109/DSD57027.2022.00127

[15] Marcin Nawrocki, Pouyan Fotouhi Tehrani, Raphael Hiesgen, Jonas Mücke, Thomas C. Schmidt, and Matthias Wählisch. 2022. On the interplay between TLS certificates and QUIC performance. In *Proceedings of the International Conference on emerging Networking EXperiments and Technologies.* https://doi.org/10.1145/3555050.3569123

[16] NIST PQC Team. 2022. *PQC Standardization Process: Announcing Four Candidates to be Standardized, Plus Fourth Round Candidates.* https://csrc.nist.gov/News/2022/pqc-candidates-to-be-standardized-and-round-4 Last accessed: June 26, 2023.

[17] National Institute of Standards and Technology. 2016. *Submission Requirements and Evaluation Criteria for the Post-Quantum Cryptography Standardization Process.* https://csrc.nist.gov/CSRC/media/Projects/Post-Quantum-Cryptography/documents/call-for-proposals-final-dec-2016.pdf Last accessed: June 26, 2023.

[18] Mike Ounsworth and Massimiliano Pala. 2022. *Composite Signatures For Use In Internet PKI.* Internet-Draft draft-ounsworth-pq-composite-sigs-07. Internet Engineering Task Force. https://datatracker.ietf.org/doc/draft-ounsworth-pq-composite-sigs/07/ Work in Progress.

[19] Christian Paquin, Douglas Stebila, and Goutam Tamvada. 2020. Benchmarking Post-quantum Cryptography in TLS. In *Post-Quantum Cryptography.*

[20] Sebastian Paul, Yulia Kuzovkova, Norman Lahr, and Ruben Niederhagen. 2022. Mixed Certificate Chains for the Transition to Post-Quantum Authentication in TLS 1.3. In *Proceedings of the Conference on Computer and Communications Security.* https://doi.org/10.1145/3488932.3497755

[21] Eric Rescorla. 2018. The Transport Layer Security (TLS) Protocol Version 1.3. RFC 8446. https://doi.org/10.17487/RFC8446

[22] Kent Rochford. 2016. Request for Comments on Post-Quantum Cryptography Requirements and Evaluation Criteria. https://www.federalregister.gov/d/2016-18150 Last accessed: June 26, 2023.

[23] Peter Williston Shor. 1994. Algorithms for quantum computation: discrete logarithms and factoring. In *Proceedings of the Annual Symposium on Foundations of Computer Science.* https://doi.org/10.1109/SFCS.1994.365700

[24] Dimitrios Sikeridis, Panos Kampanakis, and Michael Devetsikiotis. 2020. Assessing the overhead of post-quantum cryptography in TLS 1.3 and SSH. In *Proceedings of the International Conference on emerging Networking EXperiments and Technologies.* https://doi.org/10.1145/3386367.3431305

[25] Dimitrios Sikeridis, Panos Kampanakis, and Michael Devetsikiotis. 2020. Post-Quantum Authentication in TLS 1.3: A Performance Study. *IACR Cryptol. ePrint Arch.* 2020 (2020), 71.

[26] Markus Sosnowski, Wiedner Florian, Eric Hauser, Lion Steger, Dimitrios Schoinianakis, Sebastian Gallenmüller, and Georg Carle. 2023. *The Performance of Post-Quantum TLS 1.3: Raw Data.* https://doi.org/10.14459/2023mp1725057

[27] Markus Sosnowski, Florian Wiedner, Eric Hauser, Lion Steger, Dimitrios Schoinianakis, Sebastian Gallenmüller, and Georg Carle. 2023. PQS TLS Measurements repository. https://doi.org/10.5281/zenodo.10054882 https://github.com/tumi8/pqs-tls-measurements.

[28] Markus Sosnowski, Florian Wiedner, Eric Hauser, Lion Steger, Dimitrios Schoinianakis, Sebastian Gallenmüller, and Georg Carle. 2023. *PQS-TLS-Measurements website.* https://tumi8.github.io/pqs-tls-measurements Last accessed: October 25, 2023.

[29] Douglas Stebila, Scott Fluhrer, and Shay Gueron. 2022. *Hybrid key exchange in TLS 1.3.* Internet-Draft draft-ietf-tls-hybrid-design-05. Internet Engineering Task Force. https://datatracker.ietf.org/doc/draft-ietf-tls-hybrid-design/05/ Work in Progress.

[30] Stephen Hemminger. 2011. *tc-netem(8) — Linux manual page.* https://man7.org/linux/man-pages/man8/tc-netem.8.html Last accessed: September 29, 2023.

[31] Iraklis Tzinos, Konstantinos Limniotis, and Nicholas Kolokotronis. 2022. Evaluating the performance of post-quantum secure algorithms in the TLS protocol. *Journal of Surveillance, Security and Safety* (2022).

[32] David Warburton. 2021. *The 2021 TLS Telemetry Report.* https://www.f5.com/labs/articles/threat-intelligence/the-2021-tls-telemetry-report Last accessed: June 26, 2023.

[33] Bas Westerbaan and Cefan Daniel Rubin. 2022. *Defending against future threats: Cloudflare goes post-quantum.* https://blog.cloudflare.com/post-quantum-for-all/ Last accessed: June 26, 2023.

[34] Dongzhu Xu, Anfu Zhou, Xinyu Zhang, Guixian Wang, Xi Liu, Congkai An, Yiming Shi, Liang Liu, and Huadong Ma. 2020. Understanding Operational 5G: A First Measurement Study on Its Coverage, Performance and Energy Consumption. In *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication.* https://doi.org/10.1145/3387514.3405882

A ADDITIONAL MATERIAL APPENDIX

To supplement our analysis from Section 5.4 we show the measurement results obtained by emulating different network scenarios in Tables 5a and 5b. We argue it provides valuable insights to deal with such network environments and to optimize the TLS handshake performance of servers. Notable are the additional RTTs visible in the High-Delay scenario caused by the handshake size exceeding the initial TCP Congestion Window as discussed in Section 5.4. Note that all PQC algorithms have large key sizes and combining PQC KA and SA will have an additive effect on the handshake size, resulting in additional RTTs for even more algorithms.

Table 4: Median handshakes latency measured for different network scenarios. Loss, RTT, and bandwidth were emulated with netem. The width of the bars are scaled depending on the Maximum latency per table and scenario.

(a) KAs combined with `rsa:2048` as SA

Lvl	KA	No Emulation	High Loss (10%)	Low Bandwidth[1]	High Delay (1s RTT)	LTE-M[2]	5G[3]
1	**X25519**	1.77	2.03	14.11	1002.19	214.13	46.20
	bikel1	3.08	6.16	26.97	1006.56	226.99	50.61
	hqc128	1.78	2.05	51.29	1002.22	251.31	46.31
	kyber512	2.00	2.26	20.00	1002.48	220.02	46.44
	kyber90s512	2.00	2.25	19.97	1002.41	220.02	46.41
	p256	1.86	1.98	14.37	1002.20	314.70	46.18
	p256_bikel1	3.03	6.86	27.49	1007.03	227.52	51.06
	p256_hqc128	1.85	2.56	51.83	1002.75	251.88	46.78
	p256_kyber512	2.31	2.50	20.51	1002.61	220.55	46.67
3	bikel3	6.48	16.28	45.08	1016.51	508.81	60.54
	hqc192	1.94	2.95	89.25	1003.00	568.28	47.17
	kyber768	2.08	2.29	22.56	1002.44	222.57	46.46
	kyber90s768	1.99	2.30	22.56	1002.48	222.58	46.45
	p384	5.73	8.15	17.17	1008.35	245.19	52.32
	p384_bikel3	12.26	23.26	52.55	1023.41	257.99	67.36
	p384_hqc192	6.73	9.82	92.90	1009.95	507.41	54.00
	p384_kyber768	5.90	8.33	25.98	1008.51	226.12	52.52
5	hqc256	2.65	4.42	139.84	2004.79	706.85	92.77
	kyber1024	2.07	2.30	26.93	1002.52	226.95	46.51
	kyber90s1024	2.04	2.28	26.93	1002.47	367.57	46.45
	p521	12.29	17.35	21.25	1017.55	221.46	61.57
	p521_hqc256	16.91	20.83	158.08	2020.86	498.69	108.89
	p521_kyber1024	12.50	17.66	34.45	1017.79	234.60	61.81

(b) SAs combined with `X25519` as KA

Lvl	SA	No Emulation	High Loss (10%)	Low Bandwidth[1]	High Delay (1s RTT)	LTE-M[2]	5G[3]
	rsa:1024	1.03	1.37	10.99	1001.64	411.23	45.57
	rsa:2048	1.77	2.01	14.11	1002.17	214.13	46.16
1	falcon512	1.44	1.77	25.93	1001.93	225.96	45.97
	p256_falcon512	1.74	2.60	28.04	1002.78	228.05	46.81
	p256_sphincs128	15.77	15.99	297.80	2006.62	908.87	136.12
	rsa:3072	3.74	4.02	17.18	1004.17	475.42	48.17
	rsa:4096	7.19	7.51	20.42	1007.71	220.39	51.92
	sphincs128	15.32	15.53	294.72	2005.75	906.08	94.00
2	dilithium2	1.22	1.70	58.66	1001.93	258.69	45.83
	dilithium2_aes	1.17	1.55	58.69	1001.74	258.68	45.74
	p256_dilithium2	1.64	2.43	60.59	1002.66	440.73	46.64
	rsa3072_dilithium2	4.11	4.44	69.26	1004.61	354.71	48.71
3	dilithium3	1.31	1.89	78.86	1002.01	475.77	46.13
	dilithium3_aes	1.24	1.70	78.81	1001.84	278.84	45.94
	p384_dilithium3	4.23	8.97	85.28	1009.18	285.33	53.15
	p384_sphincs192	29.06	29.75	613.66	3008.62	1722.59	184.76
	sphincs192	24.13	24.55	607.69	3005.05	1473.52	181.56
5	dilithium5	1.46	2.27	106.53	2001.72	504.71	89.82
	dilithium5_aes	1.35	1.93	106.38	2001.60	480.44	89.69
	falcon1024	2.23	2.49	43.73	1002.77	445.76	46.70
	p521_dilithium5	7.90	18.01	116.98	2009.55	510.47	97.55
	p521_falcon1024	9.10	18.15	55.41	1018.15	255.46	62.14
	p521_sphincs256	61.06	61.59	857.28	4013.40	1999.60	233.52
	sphincs256	49.80	50.19	846.05	4005.62	1885.13	226.03

Legend: **Pre-Quantum** □ Full handshake (CH → Client Finished)

[1] 1 Mbit/s

[2] 10% loss, 200 ms RTT, and 1 Mbit/s bandwidth. These are realistic values for a communication over 15 km taken from Ref. [11].

[3] 4% loss, 44 ms RTT, and 880 Mbit/s. The parameters were taken from Ref. [34] and represent actually measured values.

B ARTIFACT APPENDIX

B.1 Abstract

A major goal of our research is the creation of reproducible experiments and publication of associated measurement data. This section provides an overview of the artifacts and published data. We prepared a website [28] presenting additional results and a step-by-step guide to reproduce our findings. For our experiments we used 3 nodes with optical splitters and hardware timestamping to reduce external effects; however, which is not accessible to everyone. To make the artifacts available to a broad community, we converted our scripts to run in a containerized 2-node environment, accessible via GitHub [27]. It enables executing similar measurements on any hardware; although, results might differ because of the containers, virtualized networks, software timestamping, and the different underlying hardware. However, it should reproduce the presented trends and our conclusions. Additionally, we provide raw PCAPs and CPU profiler results recorded on our infrastructure [26] to reproduce the exact paper evaluations. Artifact requirements are Debian Bullseye, Docker, and Python3. The key results are all presented data from the paper. The Docker setup does not allow profiling the CPU from within a container. Therefore, we provide the recordings of our bare-metal measurement runs.

B.2 Artifact check-list (meta-information)

- **Data set**: 161 GB, available at [26]
- **Run-time environment**: Linux kernel 5.9, Python3, and Docker
- **Output**: CSV tables, PDF plots, and PCAPs

- **How much disk space required?** 300 GB
- **Time needed to prepare workflow?** 30 min
- **Time needed to complete experiments?** 24 h
- **Publicly available?**: Yes
- **Code and data licenses**: CC BY 4.0
- **Archived**: Yes, over mediaTUM [26] and Zenodo [27]

B.3 Description

B.3.1 How to access. We provide the following resources:

- Code repository to execute experiments and evaluate results [27].
- A dataset, accessible via mediaTUM [26].
- A website with additional results and detailed descriptions [28].

B.3.2 Software dependencies. The minimal requirements is an installed Docker (we used version 24.0.6) and Python3 with additional libraries; *i.e.*, provided by the Debian packages `python3`, `python3-click`, and `python3-yaml`. The published Perf recordings can be only evaluated using a Linux Kernel Version 5.9, *e.g.*, provided by Debian Bullseye.

B.3.3 Data sets. Datasets for reproducing the exact paper results are available under [26]. It can be downloaded via `rsync` from:

```
rsync://m1725057@dataserv.ub.tum.de/m1725057/
```

B.4 Installation

After installing the dependencies, clone the repository and download our measurement results.

B.5 Experiment workflow

We suggest two workflows:

i) Run the provided experiments locally and evaluate the results. There might be differences to the paper results and not all experiments are available in the docker environment.

ii) Download our raw measurement results and evaluate them locally to reproduce the exact paper results.

B.6 Experiments Naming Schema

In the following sections, we use short names to describe the different conducted experiments. The same naming schema was used in this description, the published scripts, and measurement data. Unless stated otherwise, the experiment was conducted with our optimized OpenSSL version.

- `all-kem`: Combining all KAs together with `rsa:2048` as SA.
- `all-sig`: Measuring all SAs together with `X25519` as KA.
- `all-[kem,sig]-scenarios:` same as `all-kem` or `all-sig` but for each emulated constrained environment.
- `level[1,3,5]`: Every combination of SA and KA on the respective NIST security levels (grouped level one and two).
- `level[1,3,5]-nopush`: Same as above but conducted with the default OpenSSL behavior.
- `level[1,3,5]-perf`: Same as the `level[1,3,5]` experiments, but with additional CPU profiling.
- `all-sphincs`: Only used to identify the fastest SPHINCS+ variant.

B.7 Local Post-Quantum TLS Experiments

To run the dockerized experiments locally, we provide a script that takes a list of pre-defined experiments as argument and then creates a new folder under the `$OUT` directory per experiment:

```
./experiment.py -o \$OUT all-kem all-sig
```

Only `all-kem`, `level[1,3,5]`, `all-kem-scenarios`, `all-sig` and `all-sig-scenarios` are defined for the docker environment. The script spawns two containers for each experiment and conducts measurements for each combination of parameters in the respective configuration file of the `loop_var` directory.

B.8 Evaluation and expected results

To reproduce our evaluations, we provide an `evaluate` script that takes a list of experiment folders as argument and saves the results under `$EVAL`, e.g.:

```
./evaluate.py -o $EVAL $OUT/*
```

For each folder, the script will load the PCAPs into a PostgreSQL database, run evaluation scripts, and process the created CSVs further to retrieve the data presented in the paper.

Table 2. To reproduce the table, evaluate `all-kem` and `all-sig`. The relevant content can be found in `latencies.csv`. The columns for the bars are `partAMedian` and `partBMedian`, which are the latencies from CH to SH and from SH to Client Finished, respectively.

Figure 3. This figure is a combination of `level[1,3,5]`, and `level[1,3,5]-nopush`. The deviation analysis can be exported with the additional `--deviation-analysis True` flag.

This will create a `deviations.csv` and corresponding plot in the output directories. Figures 3a and 3b are a combination over the different security levels, skipping hybrids. Figures 3c was created by subtracting the expected latency of the `*-nopush` experiments with the ones obtained using the optimized OpenSSL version.

The `*-nopush` experiments can be run locally using a different branch of our OpenSSL library; *i.e.*, by using `basic-sphincs` instead of `basic-sphincs-psh` (*cf.*, Appendix B.9).

Table 3. The CPU profiling analysis can be reproduced by analyzing the `level[1,3,5]-perf` experiments. These results contain additional CPU profiling data, which can be transformed into our white-box analysis by passing an additional `--cpu-profiling True` flag to evaluation script.

Figure 4. The ranking is based on the `all-kem` and `all-sig` experiments. We combined the rankings from both experiments found in the `handshake_latency.pdf` plots.

Table 4. This table contains the data from the constrained environment experiments `all-[kem,sig]-scenarios`. It shows the median handshake latency (the `partAllMedian` column from `latencies.csv`) for each of KA, SA, and network emulation.

B.9 Experiment customization

Additional experiments are defined by adding files to the `loop_var` directory. Adapting the `docker-compose.yml` allows changing the OpenSSL version, set a different measurement time (we used 60 s), or change the number of parallel jobs running during the evaluation. The latter can be decreased if the PostgreSQL database takes up too much resources during evaluation.