Two Post-Quantum Signature Use-cases: Non-issues, Challenges and Potential Solutions*

Panos Kampanakis† Dimitrios Sikeridis‡

November 4, 2019

Abstract

The recent advances and attention to quantum computing have raised serious security concerns among IT professionals. The ability of a quantum computer to efficiently solve (elliptic curve) discrete logarithm, and integer factorization problems poses a threat to current public key exchange, encryption, and digital signature schemes. Consequently, in 2016 NIST initiated an open call for quantum-resistant crypto algorithms. This process, currently in its second round, has yielded nine signature algorithms for possible standardization. In this work, we are evaluating two post-quantum signature use-cases and analyze the signature schemes that seem most appropriate for them. We first consider Hash-Based Signatures for software signing and secure boot. We propose suitable parameters and show that their acceptable performance makes them good candidates for image signing. We then evaluate NIST candidate post-quantum signatures for TLS 1.3. We show that Dilithium and Falcon are the best available options but come with an impact on TLS performance. Finally, we present challenges and potential solutions introduced by these algorithms.

1 Introduction

Modern digital signature algorithms that are part of a public key infrastructure (PKI) rely on the hardness of the integer factorization (IF) and elliptic curve discrete logarithm (ECDL) problems. Assuming the existence of a large-scale quantum computer (QC), such problems can be solved in polynomial time due to Shor's quantum algorithm [36, 38]. Thus, all algorithms that rely on those problems would be vulnerable, and the majority of public key cryptography would be broken (e.g., PKI, software signing, encrypted tunnels), if a QC was available.

Evidently, due to the recent advances of quantum computing, the standardization of quantum-resistant public key algorithms is time-critical. Along these lines, US NIST has initiated a public project to standardize post-quantum (PQ) public key encapsulation and signature mechanisms, while ETSI has formed a Quantum-Safe Working Group [13] to make real-world deployment proposals. In accordance, the IETF has received proposals that introduce, and study PQ schemes in protocols [14, 19, 32, 42, 44]. Currently, the NIST project is in its second round where nine PQ signature algorithms, and 17 key exchange schemes are studied for possible standardization [1]. However, the actual integration of such schemes in PKI protocols and use-cases can be proven

*Presented at the 7^{th} ETSI/IQC Quantum Safe Cryptography Workshop 2019, Seattle, WA, USA

†Security & Trust Organization, Cisco Systems, email: pkampana@cisco.com

‡Department of Electrical and Computer Engineering, The University of New Mexico, email: dsike@unm.edu

challenging for today's Internet due to the key size overhead, and significant latency related to the heavier computational performance of these schemes.

Industry research teams from Google, Cloudflare, Microsoft, and NXP Semiconductors are actively studying the impact of PQ schemes on TLS [23–25, 27]. These efforts focus mainly on evaluating the impact of PQ key exchange (not authentication) on TLS because of its urgency; decryption of stored communications can take place after a QC is available, while impersonation cannot. We, on the other hand, focus on authentication because PKI refresh cycles are long and algorithm migration can take years (ECDSA was standardized in 2005 [3] but its adoption was still not broad a decade later).

We also study PQ Hash-Based signatures (HBS) for image signing and secure boot. In secure boot, there are two or three signature verifications as the booting process is passed from boot 0 code to a bootloader and then the OS. The signatures that are validated in each step are usually classical RSA signatures which would not be quantum-secure. Given that the boot process is amenable to short delays, we study efficient HBS parameters that would introduce quantum resistance to the secure boot.

The main **contributions of our work** include:

- We establish sample HBS parameter sets suitable for different software signing use-cases.
- We analyze their impact for software signing and show that it is immaterial on the verifier and acceptable on the signer.
- We identify two good signatures candidate algorithms from NIST's Round 2 list for TLS 1.3 authentication and compare them to RSA3072.
- We establish secure tunnel use-cases where these algorithms could be integrated relatively easily. In addition, we present use-cases where they should be carefully considered.
- We discuss challenges introduced by these algorithms and potential solutions that will optimize permanent adoption into TLS.

The rest of this work is organized as follows. Section 2 summarizes the PQ signature algorithms of interest. Section 3 discusses the experiments and findings of using HBS for software signing. Section 4 presents the data and evaluation of PQ signatures in TLS 1.3. Section 5 concludes this paper and

2 PQ candidate signature algorithms

2.1 Hash-Based Signatures

The Hash-Based Signature (HBS) family of algorithms relies on one or few-time-signature (OTS/FTS), and Merkle trees used with secure cryptographic hash functions. The first related scheme was introduced in the late 1970s [6], and it is known as the Merkle signature scheme (MSS). Using Merkle trees with hash functions and OTS/FTS to generate signatures is a reliable, and well-understood process. HBS schemes generate keypairs for the OTS/FTS. The OTS/FTS signs a message and the corresponding public key becomes a leaf in the Merkle tree. The public key is the root of the resulting Merkle tree. An HBS signature consists of the OTS/FTS signature and the path to the

Signature Scheme	Public Key Size (Bytes)	Private Key Size (Bytes)	Signature Size (Bytes)
Falcon [15]	897	1281	690
Dilithium [12]	1184	2800	2044
qTesla (provable) [2]	14880	5184	2592
LUOV [5]	32	11500	239
MQDSS [8]	46	13	20854
Rainbow [10]	58144	92960	64
GeMSS [7]	352190	13440	258
Picnic [18]	33	49	34036
SPHINCS+ [4]	32	64	16976

Table 1: Post-Quantum Signature Algorithms and Key/Signature Sizes (NIST Security Level 1).

root of the tree. The signature is verified by using the OTS/FTS public key and the authentication path to construct the root of the Merkle tree.

Currently, the most mature HBS schemes are stateful LMS [29] and XMSS [21] and stateless SPHINCS$^+$, one of the NIST signature candidates [4]. A stateful HBS relies on an OTS and the signer needs to ensure that the OTS private key state is never reused. The state management requirement is an important disadvantage that has been often brought up in IETF and NIST fora [26, 40]. While stateless SPHINCS$^+$ alleviates this issue, it also produces larger signatures and reduces overall performance.

2.2 Lattice-based Schemes: Dilithium and Falcon

Table 1 shows the key and signature sizes for all NIST Round 2 PQ signature candidates at Level 1 (128-bit security). The majority of the PQ signature schemes significantly increase the sizes of keys and signatures used by in protocols. The most promising ones, that could even be considered for immediate replacement of the currently used schemes are the lattice-based ones [30, 33]. A lattice is a set of integer linear combinations of linearly independent vectors. The cryptography-related NP-hard problems regarding lattices include the closest vector, the shortest vector and the lattice basis reduction problems. Among the NIST PQ signature candidates Dilithium [11], qTesla [2], and Falcon [15] are the ones that rely on lattice-based cryptography. In this work, we focus on Dilithium, and Falcon, as the significantly larger public key of qTesla is expected to greatly affect performance especially in the context of encrypted tunneling protocols (TLS, IKEv2).

The rest of the submitted PQ signature algorithms rely on various hard problem families that include HBS (SPHINCS$^+$ [4]), multivariate quadratic equations (MQDSS [8], LUOV [5], Rainbow [10], GeMSS [7]), and zero-knowledge proofs (Picnic [18]). We evaluated all of these schemes for TLS 1.3 and saw that their signature sizes or public keys are significantly larger than conventional ones which makes them significantly less appealing. While signature and public key size may not be a big concern for image signing since images are often large, they will have a significant impact on live protocol negotiations in protocols like TLS.

3 Hash-Based Software Image Signing

We initially investigate the use of HBS for image signing. HBS signatures offer well-established security guarantees but come with relatively big signatures and slower performance. For the image signing use-cases where software is usually several megabytes, adding a 10-40KB signature will have minimal impact on the total size. Additionally, booting an image or an OS usually takes several seconds. Thus, signature verification performance at the millisecond range will be immaterial to the whole process.

Going forward in this paper, we focus on stateful LMS and SPHINCS$^+$. The metrics we are interested in are public key and signature size which could pose a challenge to the verifier. We are also interested in the verifier code in RAM and singing and verification time. We assume that a software signer will likely be more robust without significant resource constraints.

3.1 HBS Experiments

For our experiments, we picked our own HBS tree parameters. We assumed that an image signing case would not exceed 34 trillion (2^{35}) signatures. According to our design, LMS would achieve that by leveraging a two level architecture with height $h = 20$ and $h = 15$ respectively or with an HSS tree with three levels of height $h_1 = 15$, $h_2 = 10$, $h_3 = 10$. SPHINCS$^+$ can achieve 2^{35} signatures multiple ways. We generated SPHINCS$^+$ parameters for 192 and 256-bit security levels by using the Sage script that generated the SPHINCS$^+$ NIST submission parameters. The actual parameters were picked by balancing security level, performance, total signature count and signature size.

Table 2 shows the results of our experiments. We ran all our tests in a Google Cloud [39] instance with an Intel(R) Xeon(R) CPU 2.20GH with 2 cores and 7.68GB RAM. To compile our code we used gcc version 7.4.0. The tests were run 1000 times for each parameter set. Evidently, all the private and public key sizes are of negligible size. It is also clear that verifier code of a few KB would not practically impact most image signature verifiers. Additionally, we see that LMS/HSS offers smaller signatures that stay below 8KB. SPHINCS$^+$ signatures grow to almost 23KB for 128-bit PQ security level parameters with $w = 16$. 23KB signatures are small enough for most image signing use-cases. In rare scenarios where small signatures are required LMS/HSS would be preferable. In terms of signing, we observe that all signatures take less than one second which would suffice for almost all software signing use-cases where signing takes place out-of-band. In terms of verification, none of the proposed parameters exceed 7ms which is satisfactory. As expected, parameters with $W = 4$ (LMS) or $w = 16$ (SPHINCS$^+$) verify signatures significantly faster than with $W = 8$ or $w = 256$ respectively.

3.2 Discussion about HBS software image signing

We showed that HBS performance would have minimal impact on secure boot technologies today. Signers with common hardware could sign in less than one second for either scheme. Verification would take just a few milliseconds which is fast for secure boot and software signing uses.

On the other hand, stateful HBS state management is a valid concern. NIST is, at the time of this writing, working on a Special Publication document to describe the characteristics of use-cases for stateful HBS. Software signing is believed to be one of these use-cases. Even though the state can be properly managed, it would not be trivial for crypto policy authors and testers to ensure

Parameter	PK (B)	Sig (KB)	Verifier code (KB)	Sign (ms)	Verify (ms)
`LMS_SHA256_M32_H15` with `LMOTS_SHA256_N32_W8`	60	1.63	2.15	6.24	1.30
`LMS_SHA256_M32_H20` with `LMOTS_SHA256_N32_W4`	60	2.83	2.57	1.46	0.17
`HSS` $L = 3$ with `LMS256H15W4`, `LMS256H10W4`, `LMS256H10W4`	60	7.80	3.15	1.71	0.48
SPHINCS$^+$ $n = 24$, $H = 15$, $d = 3$, $k = 18$, $a = 13$ $w = 256$	48	8.30	4.41	150.9	3.20
SPHINCS$^+$ $n = 24$, $H = 20$, $d = 4$, $k = 18$, $a = 13$ $w = 16$	48	11.49	4.46	83.4	0.63
SPHINCS$^+$ $n = 24$, $H = 35$, $d = 5$, $k = 20$, $a = 12$ $w = 16$	48	13.22	4.50	96.6	0.72
SPHINCS$^+$ $n = 32$, $H = 15$, $d = 3$, $k = 19$, $a = 16$ $w = 256$	64	14.11	4.47	677.8	4.19
SPHINCS$^+$ $n = 32$, $H = 20$, $d = 4$, $k = 19$, $a = 16$ $w = 16$	64	19.58	4.54	595.6	0.80
SPHINCS$^+$ $n = 32$, $H = 35$, $d = 5$, $k = 21$, $a = 15$ $w = 16$	64	22.62	4.53	370.2	0.95

Table 2: HBS Key, Signature Sizes, Memory footprint and Performance

that the right precautions are taken by a signer to prevent state reuse. NIST may add stateful HBS in their FIPS compliant algorithms for certain uses, but it is not clear how testing labs would test and ensure that the state is properly protected. If a testing lab cannot verify proper state management, then another concern could be raised about how could a verifier trust that none of the signatures issued by a vendor used the same state. Ut remains to be seen whether these signatures will be added in the FIPS-approved list and how state management will be attested to, but until then, stateless SPHINCS$^+$ seems to offer a more straightforward solution at the cost of worse performance and bigger signatures. Verifiers that cannot use such signatures because of memory and performance constraints will inevitably need to evaluate stateful HBS.

A smooth transition to the new PQ signatures is also an important consideration. A vendor that switches to a new signature scheme will still have a lot of deployed devices that use classical RSA signatures. Thankfully, PKCS#7 [22] allows for including multiple signatures in a `SignedData` content type. It is common for dual signatures to be included with an image, a feature that is supported by many code signing vendors. Signers that add HBS signature support could include these signatures along with classical RSA ones in signed images. A verifier that does not support HBS would still verify the RSA signature. HBS capable verifiers could verify the PQ signature.

If HBS were not chosen by the industry, eight more PQ signature candidates in NIST's Round 2 could serve as alternatives. Falcon, Dilithium and some qTesla, LUOV, MQDSS, and Picnic variants could offer PQ signature alternatives to HBS. Given that HBS is the most mature, conservative

and well-analyzed option and based on the analysis in this work, we believe that HBS is the best option for software signing and secure boot use-cases.

4 PQ Authenticated Tunnels

Next, we focus on the TLS 1.3 handshake pieces affected by PQ algorithms. We assume a classic web scenario where the client is not authenticated. The TLS 1.3 `ClientHello` message will negotiate the desired PQ signature algorithm using the `signature_algorithms` extensions. The PQ X.509 certificate/chain transmission by the server with the `ServerCertificate` message will now include PQ certificates. The server will also sign the transcripts of the handshake and transmit a PQ `CertificateVerify` message that contains a PQ signature which the client will verify along with the signatures in the certificate chain. As usual, when a certificate chain exceeds 16 KB of length, TLS will utilize Record Fragmentation to split packets before sending them [37].

A transition to PQ authentication will require new `AlgorithmIdentifier`s [9, 35] for X.509. A PQ certificate will carry the subject's PQ public key and the specific PQ signature algorithm used to create the signature. The certificate will be signed by the issuer using his PQ private key and the PQ signature is appended in the `Signature` field. The addition of the PQ public key and PQ signature to the X.509 certificate will increase the size of the certificate and thus the size of the related certificate chains (see Table 1).

4.1 PQ Authenticated TLS 1.3 Experiments

In this section we present the TLS 1.3 performance of each PQ signature scheme. To integrate PQ signatures into X.509 and TLS we utilized the `OQS OpenSSL` [35] library, which is a fork of OpenSSL that introduces post-quantum algorithms from the `liboqs` library [34, 41]. The `OQS OpenSSL` version we used was based on `OpenSSL` version `1.1.1c`.

Our experiments assume a classic web scenario where X.509 certificates are used to authenticate the server. We utilize only full 1-RTT mode TLS 1.3 handshakes with no PSK resumption and a classic `X25519` elliptic curve Diffie-Hellman (ECDH) key exchange. The PQ authentication in TLS is compared to currently used algorithms, and specifically against the 384-bit ECDSA and the 3072-bit RSA. RSA and ECDSA/EdDSA would offer 0 bits of security in a post-quantum setting. The experiments involved one local machine and several cloud-based instances [39]. The local host machine was equipped with 16 GBs of RAM, and an Intel i5-8350U processor, and the tests were implemented on a virtual machine running on four cores (1.7 GHz each) and utilizing 8GBs of RAM. Google Cloud Platform (GCP) instances were used as our remote servers, running on an Intel Xeon processor (2 cores, 2 GHz each) with 8 GBs of RAM. All instances were running Ubuntu 18.04 in an `x86_64` architecture.

We first measured TLS 1.3 handshake performance. Fig. 1 shows the TLS 1.3 handshake times for classical RSA3072, ECDSA384 and post-quantum NIST Level 1 Falcon and Dilithium with one intermediate CA in the cert chain. We show the handshake time for different distances between client and server. We can see that Dilithium and Falcon perform similarly to classic signature algorithms. Falcon's performance is worse for close distances between client and server primarily because the signing time for Falcon is ∼6ms which is noticeable for small round-trips. For higher hop counts both Falcon and Dilithium perform ∼5% slower than RSA3072 which is acceptable.

Next, we implemented a server that utilizes PQ certificates in TLS 1.3 to service multiple clients

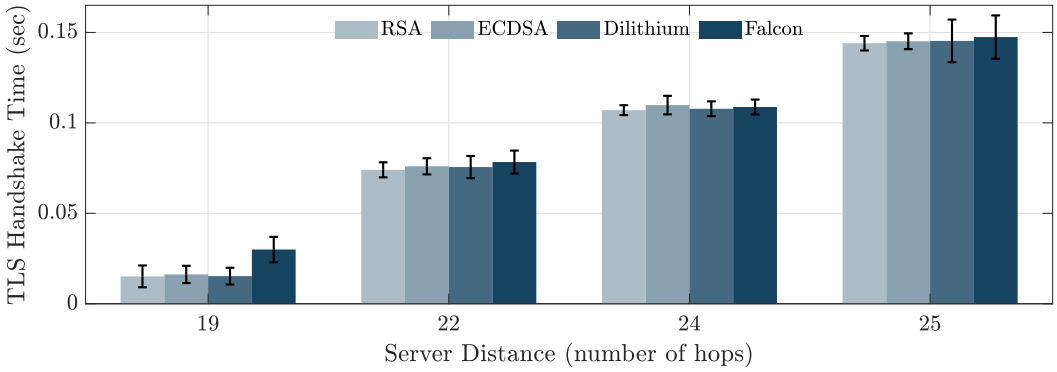

Figure 1: TLS 1.3 Handshake time per client-server distance using classic vs NIST Level 1 Dilithium & Falcon certificates

that attempt to establish secure connections. The goal was to measure the performance of the server under heavy load. To do so, an Nginx web server [31] was set up on one of the aforementioned cloud instances. We used the Siege tool [16] to simulate multiple PQ authenticated TLS connections from different clients. The client instances were located closely to each other in an attempt to emulate location based content hosting and services.

Fig. 2 shows the testing results. The figure on the left shows the completed handshakes per second with classical and PQ, NIST Level 1 signatures as the number of the simultaneous clients attempting connections increases. The figure on the right shows the failure rate with the same algorithms as the number of clients (and thus the server's load) increses. Evidently, while still at low load RSA3072 outperforms Dilithium and Falcon. At the server's saturation point, we observe that Dilithium allows the server to handle more connections per second compared to RSA3072, mainly because Dilithium signing is faster than RSA3072 at the server side. Regarding Falcon, the server's saturation point is reached early, and at a much lower transaction rate than RSA and Dilithium, which is due to Falcon's computationally heavy signing that is over 6ms.

Finally, and by utilizing the same setup, our local client was set to uniformly perform 3000 handshakes with remote servers deployed worldwide for each examined algorithm in the course of twenty-four hours. The goal was to study diverse topologies to measure TLS handshake latency observed across a longer time-frame. Table 3 summarizes the average latency in the PQ-authenticated

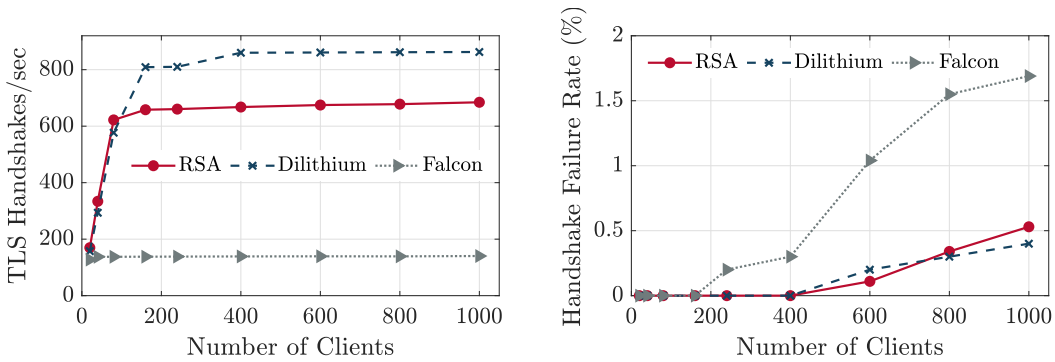

Figure 2: (Left) Handshakes/s per total clients, (Right) Handshake Failure rate per total clients

Latency (%)	Signature Algorithm		
	Falcon	Dilithium	Hash-based (SPHINCS+)
50^{th}	7.9	6.3	321.1
95^{th}	3.9	2.8	296.9

Table 3: Additional Latency of PQ NIST Level 1 Signatures in TLS Handshake over RSA at the 50^{th} and 95^{th} Percentile.

TLS handshake when compared against RSA3072 at the 50^{th} and 95^{th} percentile. The observed overhead of NIST Level 1 Dilithium was notably small making it the most pertinent for integration into TLS. Level 1 Falcon 512 also showed small overhead over RSA3072. Regarding higher security levels, the use of PQ authentication nearly doubled the handshake duration. We note that although the percentage (%) increase seems high at Level 3 and 5, the absolute time increase is ∼140s at the 50^{th} percentile and ∼235ms at the 95^{th} percentile.

4.2 Discussion about PQ Authenticated Tunnels

PQ signatures would have an impact on authenticated tunnels like (D)TLS and IKEv2/IPSec. These protocols provide fragmentation mechanisms to allow for lengthy certificate chains, but as we showed above, larger chains and potentially slower algorithms will impact the tunnel establishment. Applications with lower connection rates and tunnels that stay up longer will be less impacted than applications that establish short and quick connections. Thus, per connection overhead is important to be considered for the migration to PQ signature algorithms.

VPN tunnels usually stay up for long periods of time. For the vast majority of **IKEv2/ IPSec VPN applications**, a connection establishment that would take an extra ∼1s would not have a material impact on the tunnel. Similarly, WebVPN applications establish a TLS control connection and subsequently established data DTLS connections over a duration of a few seconds. Onward, these tunnels stay up for long sessions that usually last hours or more. A connection establishment slowdown of a few hundreds of milliseconds will not impact these connections. Thus, most PQ signature algorithms (including Dilithium and Falcon) are not likely to be a significant overall performance concern for VPN tunnels. The caveat with such use-cases is that the head-end that terminates these connections could be affected by slow performing sign/verify operations. Dilithium seems to be a better candidate than Falcon in that regard.

Web connections perform differently since they are usually fetched using short-lived connections [20]. A few KB of extra authentication data per TLS connection has low amortization over 1-2KB of actual web content [20]. In summary of our testing, we found that NIST Level 1 Dilithium and Falcon could be deployed in Web TLS X.509 certs without detrimentally impacting time-sensitive TLS applications. Dilithium is preferable over Falcon, at least until further optimizations or hardware offloading is available. Note that Level 1 Dilithium and Falcon claim a classical security level of ∼100 and ∼114 bits, which are below today's 128-bit status-quo. At higher security levels, Dilithium and Falcon could still be used if applications were amenable to approximately double the TLS handshake time or if mechanisms like [43] were widely deployed.

If Dilithium and Falcon are not standardized, tunnelling protocols will be significantly affected by the rest of the PQ signature algorithms. Given that the industry is constantly striving for faster handshakes and better performance [17, 43], it is unlikely that the impact of the PQ signature

algorithms (excluding Dilithium and Falcon) will be acceptable. In that case, more drastic changes may be necessary. These could include intermediate CA cert caching and suppression [43], adjustments of the protocol to use KEMs for authentication, use of central infrastructure (e.g., DNS or CT logs [28]) to provide the certificates offline. We consider such changes less straightforward and drastic accounting for how related protocols work today.

5 Conclusion

In this work, we evaluated the impact of using post-quantum hash-based signatures for software signing. We introduced parameter sets at different security levels that would be useful for different software signing use-cases and experimentally showed that the impact of switching to such signatures will be insignificant for the verifier compared to conventional RSA used today. We also showed that the signer will be more impacted but still at an acceptable level and we discussed practical issues with migrating the HBS signatures and their alternatives.

We also integrated PQ signature algorithms into TLS 1.3 and evaluated the TLS handshake latency observed by a client, along with the throughput of a PQ authenticated server by considering realistic network conditions. We proved that signature and certificate chain size have some impact on the total handshake time. Our results reveal that for time-sensitive protocols like HTTPS, the PQ algorithms that present the best performance are Dilithium and Falcon. Other protocols that do not require frequent connection establishments could use one of the other PQ signature algorithms as well. We showed that even slightly slower sign operations could significantly impact the total throughput of a server. However, as optimizations and hardware accelerations improve signing performance, we expect signature and key size to have the most impact on the handshake and total server throughput.

Acknowledgments

The authors would like to thank Scott Fluhrer for his useful feedback and LMS code. Additionally, special thanks to Joost Rijneveld for his feedback and comments regarding SPHINCS$^+$ parameters and the SPHINCS$^+$ implementation. We would also like to acknowledge Luke Valenta from Cloudflare for his useful feedback and experimental results regarding longer TLS records in TLS handshakes. Finally, thank you to D. Stebila and the whole OQS OpenSSL [35] team for providing a library that made our testing possible.

References

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

[2] E. Alkim, P. S. L. M. Barreto, N. Bindel, P. Longa, and J. E. Ricardini. the lattice-based digital signature scheme qtesla.

[3] ANSI. ANSI X9.62, Public Key Cryptography For The Financial Services Industry: The Elliptic Curve Digital Signature Algorithm (ECDSA), September 2005. American National Standards Institute, X9-Financial Services.

[4] J.-P. Aumasson, D. J. Bernstein, C. Dobraunig, M. Eichlseder, S. Fluhrer, S.-L. Gazdag, A. Hülsing, P. Kampanakis, S. Kölbl, T. Lange, et al. SPHINCS+ - Submission to the 2nd round of the NIST post-quantum project. `https://sphincs.org/data/sphincs+-round2-specification.pdf`, 2019. Specification document (part of the submission package).

[5] W. Beullens, A. Szepieniec, F. Vercauteren, and B. Preneel. Luov: Signature scheme proposal for NIST PQC project (Round 2 version). `https://github.com/WardBeullens/LUOV/blob/master/Supporting_Documentation/luov.pdf`, 2018.

[6] J. A. Buchmann, D. Butin, F. Göpfert, and A. Petzoldt. Post-quantum cryptography: state of the art. In *The New Codebreakers*, pages 88–108. Springer, 2016.

[7] A. Casanova, J.-C. Faugere, G. Macario-Rat, J. Patarin, L. Perret, and J. Ryckeghem. *GeMSS: A Great Multivariate Short Signature*. PhD thesis, PhD thesis, UPMC-Paris 6 Sorbonne Universités, 2017.

[8] M.-S. Chen, A. Hülsing, J. Rijneveld, S. Samardjiska, and P. Schwabe. MQDSS specifications. `http://mqdss.org/specification.html`.

[9] E. Crockett, C. Paquin, and D. Stebila. Prototyping post-quantum and hybrid key exchange and authentication in tls and ssh. Cryptology ePrint Archive, Report 2019/858, 2019. `https://eprint.iacr.org/2019/858`.

[10] J. Ding, M.-S. Chen, A. Petzoldt, D. Schmidt, and Y. Bo-Yin. Rainbow - Algorithm Specification and Documentation. `https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions`, 2019. The 2nd Round Proposal.

[11] L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, P. Schwabe, G. Seiler, and D. Stehlé. Crystals-dilithium: A lattice-based digital signature scheme. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 238–268, 2018.

[12] L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, P. Schwabe, G. Seiler, and D. Stehlé. CRYSTALS-Dilithium Algorithm Specifications and Supporting Documentation. `https://pq-crystals.org/dilithium/resources.shtml`, 2018. Submission to round 2 of the NIST post-quantum project.

[13] ETSI. ETSI TC Cyber Working Group for Quantum-Safe Cryptography. `https://portal.etsi.org/TBSiteMap/CYBER/CYBERQSCToR.aspx`, 2017. Web page. Accessed 2019-07-25.

[14] S. Fluhrer, D. McGrew, P. Kampanakis, and V. Smyslov. Postquantum Preshared Keys for IKEv2. Internet-Draft draft-ietf-ipsecme-qr-ikev2-08, Internet Engineering Task Force, Mar. 2019. Work in Progress.

[15] P.-A. Fouque, J. Hoffstein, P. Kirchner, V. Lyubashevsky, T. Pornin, T. Prest, T. Ricosset, G. Seiler, W. Whyte, and Z. Zhang. Falcon: Fast-Fourier lattice-based compact signatures over NTRU. https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions, 2018. Specification v1.1.

[16] J. Fulmer. Siege HTTP regression testing and benchmarking utility. https://www.joedog.org/siege-home/, 2019. Web page. Accessed 2019-02-09.

[17] A. Ghedini and V. Vasiliev. TLS Certificate Compression. Internet-Draft draft-ietf-tls-certificate-compression-05, Internet Engineering Task Force, Apr. 2019. Work in Progress.

[18] Z. Greg et al. The Picnic Signature Algorithm Specification. https://github.com/microsoft/Picnic/blob/master/spec/spec-v2.1.pdf, 2019.

[19] P. E. Hoffman. The Transition from Classical to Post-Quantum Cryptography. Internet-Draft draft-hoffman-c2pq-05, Internet Engineering Task Force, May 2019. Work in Progress.

[20] http archive. Trends. http://httparchive.org/trends.php.

[21] A. Huelsing, D. Butin, S.-L. Gazdag, J. Rijneveld, and A. Mohaisen. XMSS: eXtended Merkle Signature Scheme. RFC 8391, May 2018.

[22] B. Kaliski. PKCS #7: Cryptographic Message Syntax Version 1.5. RFC 2315, Mar. 1998.

[23] K. Kwiatkowski. Towards Post-Quantum Cryptography in TLS, June 2019.

[24] A. Langley. CECPQ1 results, Nov. 2016.

[25] A. Langley. CECPQ2, Dec. 2018.

[26] A. Langley. Email thread: Proposed addition of hash-based signature algorithms for certificates to the LAMPS charter, 2018. https://mailarchive.ietf.org/arch/msg/spasm/PgzLjPcg-jfywQFQs9gMLFcgRd8.

[27] A. Langley. Post-quantum confidentiality for TLS, Apr. 2018.

[28] B. Laurie, A. Langley, and E. Kasper. Certificate Transparency. RFC 6962, June 2013.

[29] D. McGrew, M. Curcio, and S. Fluhrer. Leighton-Micali Hash-Based Signatures. RFC 8554, Apr. 2019.

[30] H. Nejatollahi, N. Dutt, S. Ray, F. Regazzoni, I. Banerjee, and R. Cammarota. Post-quantum lattice-based cryptography implementations: A survey. *ACM Computing Surveys (CSUR)*, 51(6):129, 2019.

[31] Nginx. NGINX: High Performance Load Balancer Web Server and Reverse Proxy. https://www.nginx.com, 2019. Web page. Accessed 2019-02-09.

[32] M. Ounsworth and M. Pala. Composite Keys and Signatures For Use In Internet PKI. Internet-Draft draft-ounsworth-pq-composite-sigs-01, Internet Engineering Task Force, July 2019. Work in Progress.

[33] C. Peikert. A decade of lattice cryptography. *Found. Trends Theor. Comput. Sci.*, 10(4):283–424, Mar. 2016.

[34] O. Q. S. Project. liboqs. `https://github.com/open-quantum-safe/liboqs`, 2019. Web page. Accessed 2019-02-08.

[35] O. Q. S. Project. OQS OpenSSL. `https://github.com/open-quantum-safe/openssl`, 2019. Web page. Accessed 2019-02-08.

[36] J. Proos and C. Zalka. Shor's discrete logarithm quantum algorithm for elliptic curves. *Quantum Info. Comput.*, 3(4):317–344, July 2003.

[37] E. Rescorla. The transport layer security (TLS) protocol version 1.3, 2018.

[38] P. W. Shor. Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer. *SIAM J. on Computing*, 26(5):1484–1509, 1997.

[39] D. Sikeridis, I. Papapanagiotou, B. P. Rimal, and M. Devetsikiotis. A comparative taxonomy and survey of public cloud infrastructure vendors. *arXiv preprint arXiv:1710.01476*, 2017.

[40] Stateful Hash-Based Signatures - public comments on misuse resistance, 2019. `https://csrc.nist.gov/CSRC/media/Projects/Stateful-Hash-Based-Signatures/documents/stateful-HBS-misuse-resistance-public-comments-April2019.pdf`.

[41] D. Stebila and M. Mosca. Post-Quantum Key Exchange for the Internet and the Open Quantum Safe Project. Cryptology ePrint Archive, Report 2016/1017, 2016. `https://eprint.iacr.org/2016/1017`.

[42] D. Steblia, S. Fluhrer, and S. Gueron. Design issues for hybrid key exchange in TLS 1.3. Internet-Draft draft-stebila-tls-hybrid-design-01, Internet Engineering Task Force, July 2019. Work in Progress.

[43] M. Thomson. Suppressing Intermediate Certificates in TLS. Internet-Draft draft-thomson-tls-sic-00, Internet Engineering Task Force, Mar. 2019. Work in Progress.

[44] C. Tjhai, M. Tomlinson, grbartle@cisco.com, S. Fluhrer, D. V. Geest, O. Garcia-Morchon, and V. Smyslov. Framework to Integrate Post-quantum Key Exchanges into Internet Key Exchange Protocol Version 2 (IKEv2). Internet-Draft draft-tjhai-ipsecme-hybrid-qske-ikev2-04, Internet Engineering Task Force, July 2019. Work in Progress.