Impact of post-quantum hybrid certificates on PKI, common libraries, and protocols

Jinnan Fan, Fabian Willems and Jafar Zahed

Information Security Research Group,
School of Electrical Engineering and Computer Science,
University of Ottawa,
Ottawa, Ontario, Canada
Email: jfan084@uottawa.ca
Email: fabian.willems@uottawa.ca
Email: jzahe011@uottawa.ca

John Gray, Serge Mister and Mike Ounsworth*

Entrust,
Ottawa, Ontario, Canada
Email: John.Gray@entrust.com
Email: Serge.Mister@entrust.com
Email: Mike.Ounsworth@entrust.com
*Corresponding author

Carlisle Adams

Information Security Research Group,
School of Electrical Engineering and Computer Science,
University of Ottawa,
Ottawa, Ontario, Canada
Email: cadams@uottawa.ca

Abstract: In this work, we assessed the impact of post-quantum (PQ) cryptography on public key infrastructure (PKI). First, we modified a commercially available certification authority (CA) to issue 'hybrid' certificates (X.509 certificates with PQ extensions). Then we assessed the impact of using these certificates on some existing protocols, including TLS, OCSP, CMP, and EST, with open-source libraries OpenSSL and CFSSL, and with a commercially available cryptographic toolkit. We found that most of the protocols and libraries we tested worked with hybrid certificates, and some of the failures could be overcome with minor modifications to the existing software. Our work differentiates from and extends previous work by focusing on the impact of PQ algorithms on certificate issuance, revocation, and management protocols, which are necessary for enterprises to manage PKI in their environments. The impact on TLS is also investigated, allowing consistency with previous results to be evaluated.

Keywords: post-quantum cryptography; security; certification authority; certificate authority; X.509 certificates; hybrid certificates; public key infrastructure; PKI; OpenSSL; transport layer security; TLS; online certificate status protocol; OCSP; certificate management protocol; CMP; enrollment over secure transport; EST.

Reference to this paper should be made as follows: Fan, J., Willems, F., Zahed, J., Gray, J., Mister, S., Ounsworth, M. and Adams, C. (2021) 'Impact of post-quantum hybrid certificates on PKI, common libraries, and protocols', *Int. J. Security and Networks*, Vol. 16, No. 3, pp.200–211.

Biographical notes: Jinnan Fan is a PhD student in Computer Science in the School of Electrical Engineering and Computer Science, University of Ottawa. She has been working with Dr. Carlisle Adams since 2018 and collaborated with several high-tech companies, including Entrust. Her research interests include post-quantum cryptography, digital credentials and privacy. She obtained her Master's in Electrical and Computer Engineering from University of Ottawa. Her Master's thesis introduced a novel way to multi-show digital credentials, which has been nominated for an award in 2019 within the Faculty of Engineering.

Fabian Willems currently studies for a Master's in Electronic Business Technologies at the University of Ottawa, Canada. His research interests include cryptography, information privacy/security and computer security. Previously, he worked as an IT consultant and IT administrator for multiple companies in Germany. He received a Diploma in Business Administration with a focus on information technology from the University of Applied Sciences FHDW in Bergisch Gladbach, Germany.

Jafar Zahed is a Master's student studying computer science at the University of Ottawa. Having a great interest in cryptography, his master's topic focuses on finding real-world applications of homomorphic encryption. He has worked on several research projects in conjunction with the University of Ottawa, such as training IBM Watson on cyber security, creating a decryption tool for computer files infected by a particular ransomware, as well as determining the location of that ransomware's key generation algorithm, and using machine learning to distinguish APT behaviour from user behaviour on an end-user workstation. He holds a Joint Honours BSc in Computer Science and Mathematics from the University of Ottawa. He also currently works as a programmer/analyst for the Government of Canada.

John Gray is a software architect and has been with Entrust. He is actively involved in PKI development, architectural oversight and cryptographic protocol implementations. Having a keen interest in quantum computation, he is active in researching engineering implications of post-quantum cryptography and is a participating member in IETF discussions around post-quantum transition strategies for public key infrastructure (PKI). He holds a BA in Computer Science and a BMusA in Music Theory and Composition from Western University.

Serge Mister is a senior software security architect with Entrust where he focuses on the security analysis of Entrust products and researches security issues and possible solutions in existing computing environments. He has also co-authored several papers in the areas of s-box design, substitution-permutation networks, and efficient implementation of elliptic curve cryptosystems. He is currently involved in studying the impact of deploying post-quantum cryptographic algorithms, including discussions with the IETF community on the use of post-quantum algorithms in PKI environments. In the past he was involved with statistical testing of elements of the CAST design procedure and with the Advanced Encryption Standard effort. He received a Master's degree from Queen's University, Kingston, Canada after studying the cryptographic strength of the alleged RC4 stream cipher.

Mike Ounsworth is a software security architect at Entrust. His day-job is primarily application security architecture and penetration testing, with research projects in cryptography and post-quantum cryptography. He is leading discussion at IETF around post-quantum transition strategies for public key infrastructure (PKI), including primary and secondary authorship on several internet drafts. He holds an MSc in Computer Science in Robotics and Artificial Intelligence from McGill University, and an undergraduate degree in Computer Science with concentrations in mathematics and physics from Queen's University.

Carlisle Adams is a Professor in the School of Electrical Engineering and Computer Science at University of Ottawa. Prior to his academic appointment in 2003, he worked for 13 years in industry (Nortel, Entrust) in the design and standardisation of several cryptographic and security technologies for the internet. His research interests and technical contributions span applied cryptography, security, and privacy, including the CAST family of symmetric encryption algorithms, secure protocols for PKI environments, access control in electronic networks, and privacy enhancing technologies. He is a co-author of *Understanding PKI: Concepts, Standards, and Deployment Considerations*, Second Edition, Addison-Wesley in 2003.

1 Introduction

1.1 Motivation

Advances in quantum computing have raised concerns about whether the industry is ready to move to post-quantum (PQ) cryptographic algorithms, particularly for public key infrastructure (PKI) use cases. Research work in this area is necessary to prepare the market to meet the challenges of a PQ world and help to move towards PQ cryptography.

We have seen in the past that even 'straight-forward' cryptographic migrations, such as SHA-1 to SHA-256, have proved to be complex, time consuming, and fraught with compatibility issues for many enterprises and site owners (SHA-256 Transition Lessons Learned, 2011). PQ algorithms represent significantly greater agility challenges due to their differences from traditional cryptography, such as their much larger key sizes. The existing PQ technologies need to be studied to foresee barriers and design elegant migration strategies.

In this paper, we study the quantum-readiness of selected PKI components. Our goal is to test the impact of particular PQ algorithms on common infrastructures, specifically focusing on the most significant change from traditional cryptography: large public keys and digital signatures. To carry out this research we employed 'hybrid' certificates as defined in an IETF Internet Draft for 'hybrid' certificates (Truskovsky et al., 2018) as this allowed for studying the impact of certificate size on various PKI protocols, using a draft standard that implements NIST's call for 'dual modes' (NIST, 2020), without needing to

modify clients or protocols. This enables us to provide information to the community on where problems may be encountered and to suggest mitigation.

1.2 Related work

Public key cryptography is a core part of many security protocols in use today, including the ubiquitous TLS and X.509 certificates. The realisation of large-scale quantum computers able to break traditional public key algorithms such as RSA and elliptic curve cryptography (ECC) is becoming increasingly likely, with one prominent researcher in 2016 estimating a '1/7 chance of breaking RSA-2048 by 2026, 1/2 chance by 2031' (Mosca, 2016) and in 2017 '1/6 chance within 10 years' (Mosca, 2018). In Quantum Threat Timeline (2019), 22 experts were asked to give their opinion on the likelihood of a significant quantum threat to public key security. In 10 years 10/22 (45%) of these experts thought there was a 30% chance or greater, in 15 years 11/22 (50%) thought there was a 50% chance or more, and in 20 years 20/22 (90%) thought there was a 50% chance or greater that a significant threat would arise. This possibility necessitates the development and adoption of PQ public key schemes (signature, encryption, and key agreement) that have sufficient strength against both quantum and classical attacks.

The design and standardisation of PQ public key algorithms is underway, including NIST's Post-Quantum Cryptography Standardization project (NIST PQC) (NIST, 2016; Chen et al., 2016), the IETF RFCs describing the XMSS Signature Scheme (Huelsing et al., 2018) and the LMS Signature Scheme (McGrew et al., 2019), and internet drafts such as Housley et al. (2019). The NIST standardisation effort is scheduled to make draft standards for several candidate algorithms available between 2022 and 2024. Twenty-six algorithm submissions (9 of which are signature algorithms) are active candidates in round 2 of the project, at the time of writing.

The integration of PQ public key algorithms into existing protocols has inspired some research as it presents several challenges:

- PQ public key signature algorithms generally have public keys and signatures that are significantly larger than traditional public key schemes.
- PQ public key algorithms may require more computation than traditional algorithms.
- The introduction of PQ public key algorithms may cause backwards compatibility issues for systems that cannot be upgraded in-place to support new cryptographic algorithms (i.e., crypto agility).
- PQ public key algorithms may not be drop-in replacements for traditional algorithms, as they may impose additional constraints, such as the concerns raised regarding dangers of stateful hash-based signatures in the responses to NIST's request for comments on stateful hash-based signatures (Marks et al., 2019), and the fact that NIST plans to standardise multiple cryptographic algorithms where each algorithm is only approved for certain use-cases (Moody, 2018).

Also, the trade-offs between public key sizes, signature sizes, computational costs, algorithm constraints, and current confidence in algorithm security make it likely that different algorithms or algorithm parameters will be chosen for different applications.

ETSI has published a whitepaper (Pecen et al., 2015) that includes a discussion of X.509 certificates, IPSec (IKEv2), TLS, S/MIME, and SSH. This work summarises how cryptography is used in the protocols, suggests what needs to be changed to make the protocols quantum-safe, and points to research on making the needed changes. An internet draft (Truskovsky et al., 2018) has been published proposing adaptations for X.509 certificates, certificate signing requests, and certificate revocation lists (CRL), that add support for PQ public keys and signatures while remaining usable by legacy software. Some of these adaptations have been deployed in prototypes (Kampanakis et al., 2018) (including support for the EST certificate enrollment protocol) and early commercial applications (Press Release: CSS and ISARA Introduce the First and Only Quantum-Safe, Full-Stack PKI, https://www.keyfactor.com/press-releases/css-and-isara-introduce-the-first-and-only-quantum-safe-full-stack-pki/). PQ key agreement schemes have also been experimentally deployed (Braithwaite, 2016; Langley et al., 2018).

The impact of increased public key and signature sizes for PQ public key signature schemes has been investigated in Bindel et al. (2017), which studied X.509 certificates, TLS, and S/MIME. The research found that the tested libraries could successfully parse certificates exceeding 1 MB in size, but despite this some popular TLS implementations could not establish connections with TLS servers having 90 kB certificates. For S/MIME, of the five implementations tested, only Mozilla Thunderbird had problems handling large certificates. Related source code has been made available by the authors. Later research Kampanakis et al. (2018) studied TLS, DTLS, QUIC, and IPSec, and focused on assessing the impact that increased protocol message size, the resulting transmission delays and increased number of network packets would have on these protocols. The research found that the increased protocol message size could be successfully handled by both the protocols and network. The experiments with the HSS signature scheme showed non-negligible performance cost to the adoption of PQ algorithms. In Crockett et al. (2019) the TLS and SSH protocols were adapted using various approaches for implementing PQ and hybrid key exchange and authentication. Furthermore, some researchers have focused on integration of PQ cryptographic implementations into open-source libraries. For example, Chang et al. (2014) presented a full PQ SSL/TLS library using publicly available parameters, created by adapting PolarSSL. Butin et al. (2017) provided a prototype integration of the XMSS hash-based signatures

(HBS) scheme with OpenSSL in order to bring PQ authentication closer to practice. They implemented support for TLS and S/MIME in OpenSSL and found that the use case of S/MIME is much more amenable to HBS than the use case of TLS.

The simplest way to use PQ algorithms with X.509 certificates would be to place PQ public keys and signatures directly in the existing certificate fields. Certificates constructed in this way ('pure PQ certificates') could only be used with applications that understand the PQ algorithms used. The PQ certificates constructed in this paper follow the IETF Internet Draft for 'hybrid' certificates (Truskovsky et al., 2018), later branded as 'ISARA Catalyst™ Agile Digital Certificate Technology' (Isara Catalyst, 2019). Hybrid certificates place PQ data[1] – public keys, algorithm identifiers, and signatures, into non-critical X.509 v3 certificate extensions. A different approach is described in an early stage IETF Internet Draft, which combines a collection of PQ and 'classical' algorithms into a single 'composite' algorithm, with individual public key and signature objects encapsulating multiple keys and signatures (Ounsworth and Pala, 2019). In this paper, we used the hybrid format because the non-critical certificate extension mechanism allows us to study the impact of PQ certificate sizes without making any source-code modifications to the software being studied other than to the certification authority (CA). Therefore, this experiment is studying the backwards compatibility impact of increased certificate size on non-PQ-aware PKI components and the ease with which PQ algorithm support can be added.

1.3 Our contributions

This paper makes the following contributions:

1 *Modifying a CA to issue hybrid certificates:* By following the IETF Internet Draft (Truskovsky et al., 2018), our modified CA is designed to be capable of issuing hybrid certificates, which contain both an RSA and a PQ public key and signature.

2 *Evaluating the backwards compatibility of various protocols using PQ hybrid certificates:* We test existing protocols using constructed hybrid certificates of varying sizes. Based on the test results, we highlight the protocols that will not require changes and also determine the maximum certificate size that will not have backwards compatibility issues with the software we tested.

Our work differentiates from and extends previous work by focusing on the impact of PQ algorithms on certificate issuance, revocation, and management protocols, which are necessary for enterprises to manage PKI in their environments. The impact on TLS is also investigated, allowing consistency with previous results to be evaluated.

Our experiments share some common ground with the experiments in Bindel et al. (2017) and Kampanakis et al. (2018). The results for Java-based TLS match the results in Bindel et al. (2017), who claim that Java completed TLS connections with extensions of size 1.3 MiB (3.4 MiB confirmed in our experiments). They also claim that OpenSSL can handle an 80 KiB extension but not a 90 KiB extension, while the largest compatible certificate size has been determined as 100 KiB in our experiments. Also, the failures of the OpenSSL TLS client observed in this work match the results in Kampanakis et al. (2018), which claims that 135 KB certificate chains with 16 Kb keys were failing at the OpenSSL client. Moreover, we determined the specific maximum certificate size (100 KiB) that the OpenSSL TLS client will accept and observe that this maximum size can be extended.

The differences between this paper and Bindel et al. (2017) and Kampanakis et al. (2018) are as follows.

Table 1 Comparisons with previous work

	Certificate format	*Real PQ extensions*	*Tested protocols*	*Maximum sizes determined*
Bindel et al. (2017)	Second certificate in extension	×	TLSv1.2, CMS, S/MIME	×
Kampanakis et al. (2018)	Truskovsky et al. (2018)	HSS	TLSv1.2, IKEv2, DTLS, QUIC	×
This paper	Truskovsky et al. (2018)	SPHINCS$^+$	TLSv1.2, OCSP, CMP, EST	✓

As shown in the table, only Kampanakis et al. (2018) and this paper follow the hybrid certificate format of Truskovsky et al. (2018) and test a real PQ algorithm. Furthermore, our experiments complement both Bindel et al. (2017) and Kampanakis et al. (2018) by evaluating other protocols (i.e., OCSP, CMP, and EST).

2 Methodology

This section will introduce the scope of this paper and the procedure of our experiments.

Generally, as our contributions, we create a proof of concept CA to issue public key certificates using PQ cryptographic algorithms, and then determine the impact of the resulting certificate size on various software components that are used in a PKI environment, standard protocols, and common libraries. If applicable, we provide recommendations of changes or workarounds for popular libraries and protocols in order to address their compatibility issues.

Specifically, the procedure of our experiments is as follows:

1 Select one PQ algorithm that is currently going through the NIST PQC competition, and select a

parameter set for that algorithm representing 'large' bandwidth for that algorithm.

2 Design and create the software components needed to prototype a CA with a PQ private key based on the selected algorithm and parameter set, which means that the CA would be able to issue and sign certificates using the CA's PQ key. The certificates created are hybrid certificates because they contain two public keys and signatures: one PQ and one conventional (e.g., RSA or ECC).

3 Once each end entity certificate has been generated, assess the impact of using that certificate on the following certificate lifecycle protocols:

- transport layer security (TLS)
- online certificate status protocol (OCSP)
- certificate management protocol (CMP)
- enrollment over secure transport (EST).

When testing, look at whether or not the certificate size impacts the ability to issue or use the certificate in various scenarios; also measure performance characteristics. Note that the clients and servers implementing these protocols were not modified, so they do not 'understand' the PQ signature or key, and will ignore them during X.509 processing because they are in non-critical certificate extensions. We are purely testing for failures and performance impacts caused by the large certificate size.

Note that in all the experiments, only the end entity certificate contained additional PQ data. Some protocols, such as OCSP and TLS, exchange chains of certificates, and implementations may have limits not just on the size of individual certificates, but on the size of the chain or message that contains the chain. Thus, for example, even if our experiment shows that a protocol works with SPHINCS$^+$, it may be that problems would arise for a long certificate chain.

The rest of the paper is organised as follows: Section 3 describes the structure of the hybrid X.509 certificates, the modification we made to the CA to generate PQ hybrid certificates, and the generated hybrid certificates used in the experiments. Section 4 describes the experimental setup and results of using the hybrid certificates over various protocols. Section 5 summarises the findings. Section 6 provides some future experimental directions.

3 Creation of hybrid certificates

3.1 SPHINCS$^+$

NIST PQC was initiated by the National Institute of Standards and Technology to solicit, evaluate, and standardise one or more PQ (quantum-resistant) public key cryptographic algorithms (NIST, 2016). Hash-based signatures are the first PQ signature algorithms seeing public adoption (Huelsing et al., 2018; McGrew et al., 2019).

SPHINCS-256 (Bernstein et al., 2015) is a high-security PQ hash-based signature scheme, which provides 128-bit security even against attackers equipped with quantum computers and can be a drop-in replacement for current signature schemes because of its stateless property. Signatures are approximately 41 KB, public keys are approximately 1 KB, and private keys are approximately 1 KB.

Figure 1 X.509 certificates today

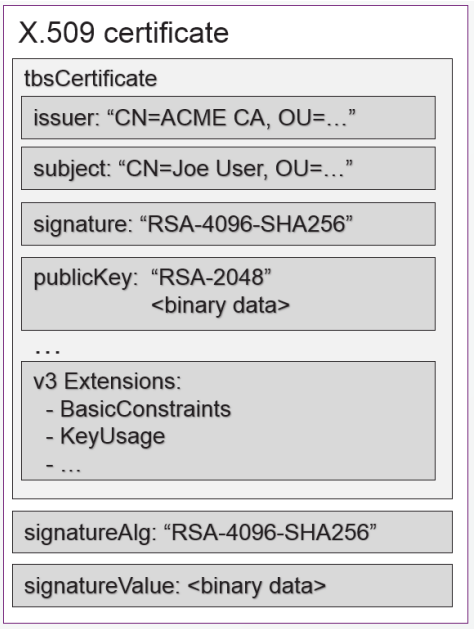
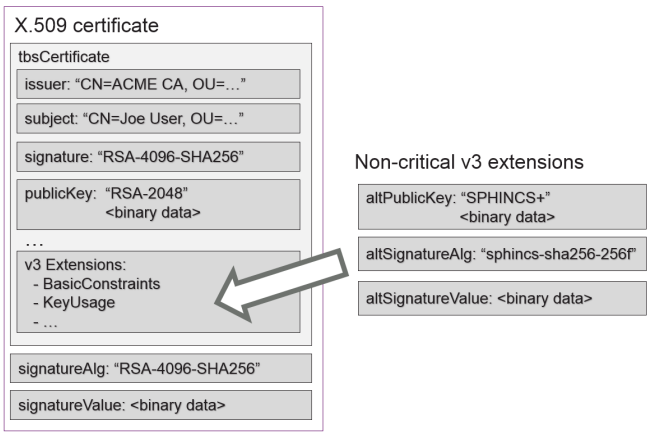


Figure 2 Hybrid certificates: X.509 certificate extension



SPHINCS$^+$ advances the SPHINCS signature scheme. Particularly, it reduces the signature size (e.g., 30 KiB).[2] SPHINCS$^+$ from round 1 of the NIST PQC was the current hash-based algorithm in the NIST competition at the beginning of our experiments.

We used one instantiation (sphincs-sha256-256f) of the round 1 version of SPHINCS$^+$ in our hybrid certificates because it is the largest SPHINCS$^+$ parameter set that uses the SHA family of hash functions. It has a public key +

signature size of 64 + 49,216 bytes. sphincs-sha256-256f is a fast ('f' for 'fast') SPHINCS$^+$ implementation based on the hash function SHA-256 for NIST security level 5 (Bernstein et al., 2017).

3.2 Hybrid certificate format

In the hybrid certificate format (Truskovsky et al., 2018), all PQ data (public keys and signatures) is in non-critical X.509 certificate extensions, so protocols can process these certificates even without supporting PQ cryptographic algorithms.

A simplified view of the format of X.509 certificates used today (which only contain a single public key and a single signature) and the hybrid certificate with non-critical X.509 certificate extensions are shown in Figures 1 and 2.

As we can see, there are three PQ extensions in the hybrid certificates:

- *Subject alt public key info:* Carries the alternative (PQ) public key bits and the algorithm identifier specifying the public key type.
- *Alt signature algorithm:* Contains the identifier for the alternative (PQ) signature algorithm used by the CA.
- *Alt signature value:* Contains the alternative (PQ) signature bits.

In the X.509 standard version 3 (Cooper et al., 2008), the body of the certificate, called a `tbsCertificate` ('tbs' for 'to be signed'), contains all the fields in Figure 1 except the last two (the conventional RSA signature by the CA over the `tbsCertificate` and its identifier). As for hybrid certificates in Figure 2, the `preTBSCertificate` is created by removing the third PQ extension, *AltSignatureValue*, and the signature field from the `tbsCertificate`. The 'inner' PQ signature (*AltSignatureValue*) will be computed over the `preTBSCertificate`. The 'outer' RSA signature will be over the `tbsCertificate` including all the PQ extensions, even the PQ signature bytes (*AltSignatureValue*).

To verify the PQ signature in the PQ extensions, the `tbsCertificate` field is extracted from the certificate and the *AltSignatureValue* extension and signature fields are removed, resulting in a `preTBSCertificate`. Then using the algorithm specified in the *AltSignatureAlgorithm* extension of the `preTBSCertificate` and the alternative public key from the CA's *SubjectAltPublicKeyInfo* extension, verify the alternative signature.

3.3 Modification of a CA

We constructed a prototype hybrid CA by modifying an existing conventional one, the entrust security manager product. The hybrid CA can generate hybrid end entity certificates and sign the end entity certificates using PQ algorithms. This modified CA uses a new external software component, which we named *pqcrypto*, to sign certificates with a PQ algorithm.

Figure 3 Hybrid certificates creation 'pipeline' (see online version for colours)

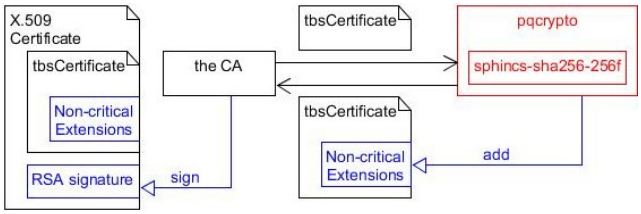


Figure 3 shows an overview of the modified CA. The creation pipeline of certificates has been modified so that immediately before signing the `tbsCertificate`, it would be transited as a byte array to *pqcrypto* which will insert the PQ extensions into the `tbsCertificate`, and then transit it back to the CA to be signed.

Within the *pqcrypto* module, we are using the reference implementation submitted to NIST, sphincs-sha256-256f (SPHINCS+ Team, 2018) which is a C-based SHA256-SPHINCS implementation.

We have implemented a Java application that can verify the PQ extensions in a hybrid certificate as described in the IETF draft (Truskovsky et al., 2018). However, for the experiments using certificates containing PQ extensions in this paper, the PQ extensions were present in the certificates, but not actually used, because we were studying the backwards compatibility of unmodified clients and servers as they exist today.

3.4 Hybrid certificates of various sizes

Our modified CA is capable of issuing the following types of hybrid certificates:

- *Conventional:* No alternative (PQ) algorithm extensions are inserted.
- *Hybrid-size-only:* The three PQ extensions are added to the generated certificates. The content of the *subject alt public key info* and *alt signature value* extensions is set to random data of a configurable size. The *alt signature algorithm* extension is set to a fixed proprietary object identifier value.
- *Hybrid-sphincs$^+$:* Three extensions are added to the generated certificates. The content of these extensions corresponds to a valid SPHINCS$^+$ signing algorithm and includes a genuine SPHINCS$^+$ signature, with a fixed proprietary object identifier value in the *alt signature algorithm* extension.

Hybrid certificates of varying sizes were produced with the modified CA in order to test the backwards compatibility and performance of applications that use various cryptographic protocols involving certificates.

The certificates we used for testing are shown in Table 2. Note that the results in the table are for NIST PQC round 1 candidates.

For each end entity certificate, the PQ extensions were generated having each of the sizes shown in the

table above. As for 'P' and 'G', each of them just contains the alternative (PQ) extensions of the respective sizes of random data, not the 'real' PQ public key and signature generated by picnicl5ur or GeMSS256. These two algorithms were chosen as they represent the largest public key size (GeMSS256) and signature size (picnicl5ur) of all NIST PQC first round candidates, which complements SPHINCS$^+$ which represents the largest hash-based signature.

Since lattice-based schemes do not represent an extreme in either public key or signature size, they were not explicitly studied, but are expected to work wherever the larger algorithms do.

Table 2 The approximate* size of the experimental hybrid certificates in bytes

	PQ public key	*PQ signature*	*Certificate*
N	N/A	N/A	1,029
S	64	49,216	50,434
P	65	209,478	210,692
G	3,603,792	104	3,605,052

Notes: N for 'certificate with no PQ extension'
S for 'a hybrid certificate with sphincs-sha256-256f extensions (real SPHINCS$^+$)'
P for 'a hybrid-size-only certificate with picnicl5ur size extensions'
G for 'a hybrid-size-only certificate with GeMSS256 size extensions'.
*different certificates were used with different protocols, and the size of different certificates with the same PQ extensions slightly varies due to the information included in the certificates (e.g., the name in the certificate, and other extensions needed for the protocol).

4 Experimental setup and results

In this section, we will use the certificates listed in Table 2 for experiments. The experimental setup and results are provided. Also, the test results are summarised in Table 3 in the next section.

4.1 TLS

The TLS protocol negotiates cryptographic parameters and uses them to establish a secure communication channel. The current version TLS 1.3 was defined in Rescorla (2018). However, at the time of writing, TLS 1.3 is still in early adoption and not all of the tested clients support it, so testing was done with TLS 1.2 (Dierks and Rescorla, 2008).

To study whether the TLS 1.2 connection between servers and clients is able to handle hybrid certificates of different sizes, we tested using a commercially available Java toolkit, Entrust Authority Security Toolkit for the Java Platform, and the OpenSSL command line tool.

1 *Java TLS:* A commercially available Java toolkit, the Entrust Security Toolkit for the Java Platform, was used on Java 1.7 64-bit to exercise the Oracle Java TLS implementation through the JSSE API. The toolkit contains two TLS samples: a simple TLS server and a simple TLS client. The results show that the TLS 1.2 handshake succeeds with all four tested sizes of hybrid certificate. Note that the time consumed for the TLS handshake to complete increases as the certificate size increases, from less than 1 second (119 ms) with no PQ extensions to 1815 ms with GeMSS256 size extensions, even though we are not invoking the PQ cryptographic algorithm. It was expected that the length of time would increase as the certificate size increased. We did not further analye the performance characteristics introduced by larger certificates, network latency and protocol overhead as these things are discussed in detail in Kampanakis et al. (2018).

2 *OpenSSL TLS:* A TLS server was started using OpenSSL 1.0.2p 14 August 2018 on Windows 8.1 Enterprise and a client was started using OpenSSL of both version[3] 1.0.2k-fips 26 January 2017 and 1.1.1b 26 February 2019 on CentOS 7.

It was observed that the server and client functionality are able to establish a TLS connection when using a real SPHINCS$^+$ extension (TLSv1.2 handshake with cipher ECDHE-RSA-AES256-GCM-SHA384). However, when using the hybrid certificate with picnicl5ur size and GeMSS256 size extensions, the connection could not be successfully established (TLSv1.2 handshake with no cipher as the client stopped establishing the connection while reading the server certificate). We were able to connect to the OpenSSL server using a Java-based TLS client.

A stock OpenSSL 1.1.1b 26 February 2019 was debugged to identify the maximum certificate size that could be used successfully with an OpenSSL client. The maximum size of TLS response accepted by the client was found to be $100 * 1{,}024 = 102{,}400$ bytes by default. The API function `SSL_CTX_set_max_cert_list`[4] can be used to allow the TLS client to connect even when a larger certificate has been used (e.g., the maximum size for TLS response could be set to 250,000 bytes to handle a single picnicl5ur size certificate, or 4,000,000 bytes to handle a single GeMSS256 size certificate, or more to handle a chain). The OpenSSL 1.1.1b 26 February 2019 TLS client was modified to call `SSL_CTX_set_max_cert_list` and re-compiled then the TLS client was re-tested. The updated client successfully connected for all four tested certificate sizes.

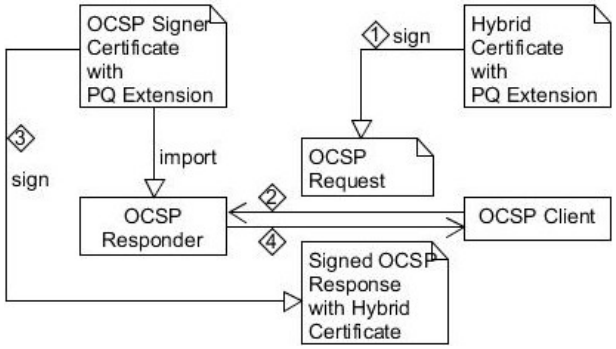
4.2 OCSP

OCSP is an internet protocol used for obtaining the current revocation status of an X.509 digital certificate without requiring CRLs, described in Santesson et al. (2013). OCSP is more likely to provide timely revocation information than with CRLs.

4.2.1 OCSP with command line tools

An overview of how we used our hybrid certificates with OCSP is shown in Figure 4. The client creates an OCSP request, which may or may not be signed with a hybrid certificate (step 1), and sends the OCSP request to the OCSP responder (step 2). Then the responder will create an OCSP response which is signed with its hybrid certificate (step 3), and send back the OCSP response to the client (step 4). All signatures in this experimental setup use only the entity's RSA key, but the messages transmitted in steps 2 and 4 contain certificates with PQ extensions, so they are larger than normal.

Figure 4 OCSP with the use of hybrid certificate



In order to study the impact of certificate size on both the OCSP client and responder, we tested the OpenSSL 1.0.2k-fips 26 January 2017/1.1.1b 26 February 2019 and CFSSL[5] 1.3.2 as command line servers on CentOS 7 while OpenSSL 1.0.2p 14 August 2018 was used as a command line client on Windows 8.1 Enterprise. We found that the OCSP servers (responders), as implemented by both tools, work for all of the four sizes of hybrid certificates. The OpenSSL server works for the OCSP requests with or without a signature signed by the client using its 'conventional' RSA key, which is stored in a hybrid certificate. In the signed case, this hybrid certificate will be sent along with the signed OCSP request to the OCSP responder. Note that the CFSSL server was only tested with OCSP requests without a signature since the OCSP server does not support signed OCSP requests.

However, the OpenSSL OCSP client functionality fails when picnicl5ur-size and GeMSS256-size extensions are returned. A generic error message 'error querying OCSP responder' was displayed by the client.

A stock OpenSSL 1.1.1b 26 February 2019 was debugged to identify the maximum certificate size that could be used successfully with an OpenSSL OCSP client. Similar to the TLS findings in Section 4.1, the maximum size of OCSP response was found to be set to 100 $*$ 1,024 = 102,400 bytes by default. We determined that the API function `OCSP_set_max_response_length`[6] allows the OCSP client to process a larger certificate (e.g., the maximum size for OCSP responses could be set to 250,000 bytes to handle a single picnicl5ur size certificate, or 4,000,000 bytes to handle a single GeMSS256 size certificate, or more to handle a chain). Then the OpenSSL 1.1.1b 26 February 2019 OCSP client was modified to include a call to `OCSP_set_max_response_length`, re-compiled and re-tested. The modified client successfully performed OCSP for all four tested certificate sizes.

4.2.2 OCSP with browsers

An overview of web browsers performing OCSP revocation checking is shown in Figure 5:

1 The web browser (client) connects to a website over the HTTPS protocol.

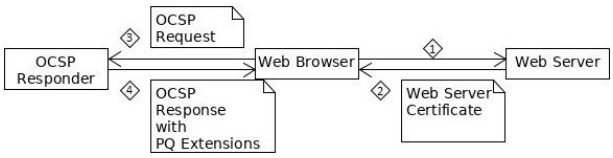
2 The web server sends back the certificate, which contains the URL (address) of the OCSP responder in the certificate's authority info access (AIA) extension.

3 The browser sends an OCSP request to the corresponding OCSP responder.

4 The responder sends back the OCSP response which is signed with one of the hybrid certificates from Table 2. Note that only the messages transmitted in this step (step 4) are larger than standard ones because of the PQ extensions.

In our study of web OCSP implementations, we tested Mozilla Firefox 66.0.3 (32-bit) and Internet Explorer (IE) 11.0.9600.19326 as the OCSP clients on Windows 8.1 Enterprise. The OpenSSL 1.1.1b 26 February 2019 command line tool on CentOS 7 was used both for the OCSP responder and for the web server hosting the web page fetched in the web browser to trigger an OCSP request. The status of the web certificate was set to 'revoked' for the purpose of identifying whether or not the OCSP response was correctly interpreted by the web browser. Because some web browsers by default ignore errors when fetching OCSP responses, if a 'valid' status had been returned, it would have been hard to determine whether or not the OCSP response was in fact correctly processed. In our test, if the OCSP response was correctly interpreted, a 'revoked' error massage would be displayed by the web browser. Before testing, the CA root certificate needed to be imported into the browsers' certificate store. Also, the OCSP caches on both Firefox and IE needed to be deleted before each test. We did not test on Google Chrome because OCSP checks are disabled by default in recent versions (Langley et al., 2012).

Figure 5 OCSP with the web browser



We found that the IE client works for all four sizes of OCSP response. However, it was observed that the Firefox functionality fails when picnicl5ur-size and GeMSS256-size extensions are returned. A generic

error message SEC_ERROR_OCSP_MALFORMED_RESPONSE was displayed by the web browser. It seems like there is a limit for the certificate size in Firefox, since the same response sizes are accepted by IE.

To identify the maximum size of OCSP responder certificate (or certificate chain) that is successfully handled by the Firefox OCSP client, a proprietary tool was used that allows real-time interception and modification of transmission control protocol (TCP) packets. Specifically, we deleted portions of different sizes in the large returned certificate extensions in real time when receiving the OCSP response, and found the size for which Firefox's error message changed from SEC_ERROR_OCSP_MALFORMED_RESPONSE to SEC_ERROR_OCSP_SERVER_ERROR (this second response likely arises because the certificate has an invalid signature due to our modification, but it shows that the certificate is now being processed). With this technique we found that $2^{16} - 1$ (65,535) bytes is the maximum size of OCSP response which the Firefox OCSP client accepts.

We also noticed by observing the HTTP messages with Wireshark[7] that IE uses an HTTP GET request, while Firefox is using an HTTP POST request. Both request methods POST and GET are allowed in Santesson et al. (2013), Appendix A1.

We observed that the OpenSSL 1.0.2k-fips 26 January 2017 can only handle the POST form of OCSP queries as documented.[7] OpenSSL 1.1.1b 26 February 2019 was found to be able to handle both of the methods, even though the documentation has not been updated.

4.3 CMP

CMP is an internet protocol used by a client system to obtain X.509 certificates from a server (i.e., a CA) in PKIs, which is described in the standards document (Adams et al., 2005).

In this experimental setup, the non-PQ-aware CMP client generates a certificate request (CMP initialisation request message) containing an RSA public key, and the CA responds with a CMP initialisation response message containing two certificates (one signing and one encryption containing a server-generated key) with the additional PQ extensions for the end entity.

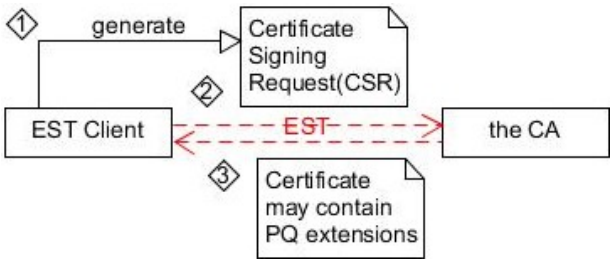
We tested with the size of the PQ extensions from 0 to GeMSS256 size, using CMP with our internal software and toolkit, Entrust Authority Security Manager and Entrust Authority Security Toolkit for the Java Platform. The results show that there were no issues with certificate creation for any of the requested sizes.

4.4 EST

EST is a simple, yet functional, CMP in PKIs, described in Pritikin et al. (2013).

We tested whether the client system can successfully receive all four sizes of hybrid certificates from Table 2 using EST from the CA, as implemented in Entrust Authority Administration Services. The main steps are shown in Figure 6. The results show that the CA can successfully issue X.509 certificates with PQ extensions, and the client can successfully receive them at all tested sizes. There was no noticeable increase in execution time, despite the increase in certificate size.

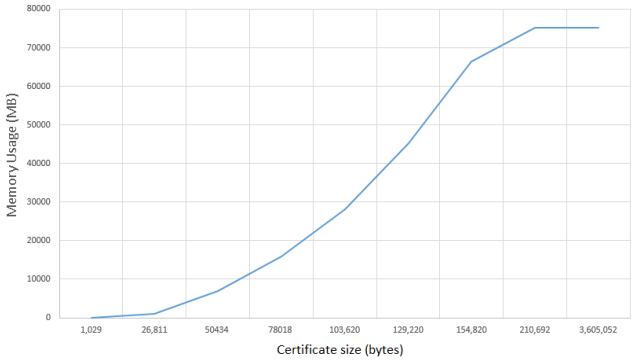
Figure 6 Obtaining X.509 certificates over EST (see online version for colours)



4.5 macOS issue

We also noticed that, on macOS High Sierra, the act of opening hybrid certificate files in the Finder application takes a significant amount of memory. As shown in Figure 7, the memory usage for rendering a certificate is quadratic ($R^2 = 0.999$) in the size of the certificate, and even a modestly-sized PQ certificate exhausted both the RAM and virtual disk space available to the system.

Figure 7 Memory usage when parsing hybrid certificates in MacOS certificate viewer (see online version for colours)



The X.509 certificate could not be displayed (viewing certificates in a macOS finder window) when the hybrid certificate had a size larger than picnicl5ur (i.e., 210,692 and 3,605,052 bytes shown in the figure), by which point the 'keychain access' process was reported to be using around 75 GB of RAM and was non-responsive.

The MacBook Pro used in this test was equipped with a 2.8 GHz Intel Core i7 processor and 16 GB of 2,133 MHz LPDDR3 memory and had approximately 104 GB of free disk space at time of testing.

5 Discussion

The experimental results for the backwards compatibility of hybrid certificates on different standard protocols, TLSv1.2,

OCSP, CMP, and EST, using different tools have been summarised in Table 3.

Table 3 Experimental results over various protocols using X.509 hybrid certificates

Protocols and toolkits			*Sizes tested*				*Failure occured at (bytes)**
			N	*S*	*P*	*G*	
TLSv1.2	Server	OpenSSL	●	●	●	●	-
		Java	●	●	●	●	-
	Client	OpenSSL	●	●	◑	◑	$100 * 1{,}024 + 1$
		Java	●	●	●	●	-
OCSP	Server	OpenSSL	●	●	●	●	-
		CFSSL	●	●	●	●	-
	Client	OpenSSL	●	●	◑	◑	$100 * 1{,}024 + 1$
		Firefox	●	●	○	○	2^{16}
		IE	●	●	●	●	-
CMP			●	●	●	●	-
EST			●	●	●	●	-

Notes: Sizes: N (1 KB), S (50 KB), P (210 KB), G (3.6 MB). Size details in Table 2.
*'-' denotes no failures at any of the tested sizes.
● success, ○ failure and ◑ failures could be overcome with a minor code change to the application to set a larger max message size.
Note that failure values are the overall size of the TLS or OCSP response message, not specifically the certificate.

We can see from the table that:

- OpenSSL's TLSv1.2 server implementation works with all tested certificate sizes; OpenSSL's TLSv1.2 client implementation by default works with hybrid certificates of the size needed for the sphincs-sha256-256f extensions (102,400 byte limit found to be set by default in the libraries) but the maximum size is able to be modified by calling an API as described in Subsection 4.1.
- Java TLSv1.2 (invoked through a Entrust Authority Security Toolkit for the Java Platform sample application) works at all tested certificate sizes.
- OCSP servers (both OpenSSL and CFSSL) work with all tested certificate sizes; IE's OCSP client works with all tested certificate sizes; Firefox's OCSP client fails for certificates larger than $2^{16} - 1 = 65{,}535$ bytes; the OpenSSL OCSP client works with hybrid certificates of the size needed for the sphincs-sha256-256f extensions (102,400 byte limit found to be set by default in the libraries) but the maximum size is able to be modified by calling an API as described in Subsection 4.2.
- CMP (Entrust Authority Security Manager and Entrust Authority Security Toolkit for the Java Platform) works with all tested certificate sizes.
- EST (in Entrust Administration Services) works with all tested certificate sizes.

In conclusion, most of the protocols and libraries we tested work smoothly with the hybrid certificates, and some of the failures can be overcome with minor modifications to the existing software.

6 Future work

This paper aims to provide guidance to the community about the impact of particular PQ algorithms, and specifically the impact of the resulting hybrid certificate sizes, on common infrastructures. In order to provide more examples of safe migration paths where they exist and recommend changes (mitigation) based on the discovered issues, here are some possible directions to expand the experiments:

- Create a publicly accessible CA that can generate demo PQ certificates for the community to test. There are several possible avenues:
  - a Extend the CA to provide pure PQ certificates instead of hybrid certificates, redoing size analysis.
  - b Extend the CA to provide composite certificates following the internet draft (Ounsworth and Pala, 2019), redoing size analysis.
  - c Extend the original certificate generation by implementing additional PQ algorithms or parameter sets (e.g., Dilithium [Ducas12019]).
  - d In addition to the end entity certificates being hybrid, the CA certificate should also be a hybrid certificate with two keys: one PQ and one conventional (e.g., RSA or ECC).
- Test other protocols and data formats, for example, Simple Certificate Enrollment Protocol (SCEP), Security Assertion Markup Language (SAML), eXtensible Markup Language (XML) Signatures, etc.
- Provide migration and implementation guidelines for protocols that require design or architecture changes in order to support PQ algorithms either alone or in a hybrid mode.
- Implement the CA verifier and client verifiers to support the verification of PQ extensions in hybrid certificates.
- Improve the measurement of execution time by applying, for example, JMeter, and determine algorithmic or implementation causes of observed slowdown that results from increased certificate size.
- Include other protocol-specific testing metrics (e.g., bandwidth, number of packets, increased congestion, etc.).
- Further study performance and compatibility impacts to TLS.

a On commonly used web browsers.

b On internet of things (IoT) devices; particularly studying the feasibility of PQ cryptography in severely memory and bandwidth contrained environments.

Acknowledgements

The authors gratefully acknowledge financial support for this work provided by MITACS through the MITACS Accelerate Program.

References

Adams, C., Farrell, S., Kause, T. and Mononen, T. (2005) *Internet X.509 Public Key Infrastructure Certificate Management Protocol (CMP)*, RFC 4210, RFC Editor, September [online] http://www.rfc-editor.org/rfc/rfc4210.txt (accessed 14 August 2019).

Bernstein, D.J., Hopwood, D., Hülsing, A., Lange, T., Niederhagen, R., Papachristodoulou, L., Schneider, M., Schwabe, P. and Wilcox-O'Hearn, Z. (2015) 'SPHINCS: practical stateless hash-based signatures', in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, Springer, Berlin, Heidelberg, April, pp.368–397.

Bernstein, D.J., Dobraunig, C., Eichlseder, M., Fluhrer, S., Gazdag, S.L., Hülsing, A., Kampanakis, P., Kölbl, S., Lange, T., Lauridsen, M. et al. (2017) *SPHINCS+ Submission to the NIST Post-Quantum Project*, December.

Bindel, N., Herath, U., McKague, M. and Stebila, D. (2017) 'Transitioning to a quantum-resistant public key infrastructure', in *International Workshop on Post-Quantum Cryptography*, Springer, pp.384–405.

Braithwaite, M. (2016) 'Experimenting with post-quantum cryptography', in *Google Securtiy Blog*, July.

Butin, D., Wälde, J. and Buchmann, J. (2017) 'Post-quantum authentication in OpenSSL with hash-based signatures', in *2017 Tenth International Conference on Mobile Computing and Ubiquitous Network (ICMU)*, IEEE, pp.1–6.

Chang, Y-A., Chen, M-S., Wu, J-S. and Yang, B-Y. (2014) 'Postquantum SSL/TLS for embedded systems', in *2014 IEEE 7th International Conference on Service-Oriented Computing and Applications*, IEEE, pp.266–270.

Chen, L., Jordan, S., Liu, Y-K., Moody, D., Peralta, R., Perlner, R. and Smith-Tone, D. (2016) *Report on Post-Quantum Cryptography*, US Department of Commerce, National Institute of Standards and Technology.

Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R. and Polk, W. (2008) *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*, RFC 5280, RFC Editor, May [online] http://www.rfc-editor.org/rfc/rfc5280.txt (accessed 14 August 2019).

Crockett, E., Paquin, C. and Stebila, D. (2019) 'Prototyping post-quantum and hybrid key exchange and authentication in TLS and SSH', in *NIST 2nd Post-Quantum Cryptography Standardization Conference 2019*, August, To Appear.

Dierks, T. and Rescorla, E. (2008) *The Transport Layer Security (TLS) Protocol Version 1.2*, RFC 5246, RFC Editor, August [online] http://www.rfc-editor.org/rfc/rfc5246.txt (accessed 14 August 2019).

Housley, R. (2019) *Use of the HSS/LMS Hash-based Signature Algorithm in the Cryptographic Message Syntax (CMS)*, Internet-Draft draft-ietf-lamps-cms-hash-sig-09, IETF Secretariat, August [online] http://www.ietf.org/internet-drafts/draft-ietf-lamps-cms-hash-sig-09.txt (accessed 14 August 2019).

Huelsing, A., Butin, D., Gazdag, S., Rijneveld, J. and Mohaisen, A. (2018) *XMSS: eXtended Merkle Signature Scheme*, RFC 8391, RFC Editor, May.

Isara Catalyst (2019) December [online] https://www.isara.com/catalyst/ (accessed 14 August 2019).

Kampanakis, P. (2018) 'Towards backward-compatible post-quantum certificate authentication', in *Cisco Blog > Security*, April.

Kampanakis, P., Panburana, P., Daw, E. and Van Geest, D. (2018) 'The viability of post-quantum X.509 certificates', *IACR Cryptology ePrint Archive*, Vol. 63.

Langley, A. (2012) 'Revocation checking and Chrome's CRL', in *ImperialViolet Blog*, February [online] https://www.imperialviolet.org/2012/02/05/crlsets.html (accessed 14 August 2019).

Langley, A. (2018) 'Post-quantum confidentiality for TLS', in *ImperialViolet Blog*, April [online] https://www.imperialviolet.org/2018/04/11/pqconftls.html (accessed 14 August 2019).

Marks, L. et al. (2019) 'Stateful hash-based signatures', in *Public Comments on Misuse Resistance*, February [online] https://csrc.nist.gov/CSRC/media/Projects/Stateful-Hash-Based-Signatures/documents/stateful-HBS-misuse-resistance-public-comments-April2019.pdf (accessed 14 August 2019).

McGrew, D., Curcio, M. and Fluhrer, S. (2019) *Leighton-Micali Hash-Based Signatures*, RFC 8554, RFC Editor, April.

Moody, D. (2018) 'Let's get ready to tumble. The NIST PQC 'competition', in *Proc. of First PQC Standardization Conference*, April, pp.11–13.

Mosca, M. (2016) *Managing the Quantum Risk to Cybersecurity*, Technical Report, University of Waterloo, Intitute of Quantum Computing, April.

Mosca, M. (2018) *Preparing for the Quantum Era*, Keynote Talk, CryptoWorks21, Waterloo, Canada, 18 August [online] https://crypto.iacr.org/2018/affevents/qsci/medias/Michele_Mosca.pdf.

NIST (2016) *Post-Quantum Cryptography Standardization*, December.

NIST (2020) *Post-Quantum Cryptography FAQs*, June.

Ounsworth, M. and Pala, M. (2019) *Composite Keys and Signatures for Use in Internet PKI*, Internet-Draft draft-ounsworth-pq-composite-sigs-01, IETF Secretariat, July [online] http://www.ietf.org/internet-drafts/draft-ounsworth-pq-composite-sigs-01.txt (accessed 14 August 2019).

Pecen, M. et al. (2015) *Quantum Safe Cryptography and Security: An Introduction, Benefits, Enablers and Challenges*, White Paper, European Telecommunications Standards Institute, June.

Press Release: CSS and ISARA Introduce the First and Only Quantum-Safe, Full-Stack PKI, April [online] https://www.keyfactor.com/press-releases/css-and-isara-introduce-the-first-and-only-quantum-safe-full-stack-pki/ (accessed 14 August 2019).

Pritikin, M., Yee, P. and Harkins, D. (2013) *Enrollment Over Secure Transport*, RFC 7030, RFC Editor, October.

Quantum Threat Timeline (2019) October [online] https://globalriskinstitute.org/publications/quantum-threat-timeline/ (accessed 14 August 2019).

Rescorla, E. (2018) *The Transport Layer Security (TLS) Protocol Version 1.3*, RFC 8446, RFC Editor, August.

Santesson, S., Myers, M., Ankney, R., Malpani, A., Galperin, S. and Adams, C. (2013) *X.509 Internet Public Key Infrastructure Online Certificate Status Protocol – OCSP*, RFC 6960, RFC Editor, June [online] http://www.rfc-editor.org/rfc/rfc6960.txt (accessed 14 August 2019).

SHA-256 Transition Lessons Learned (2011) *Federal Public Key Infrastructure Policy Authority*, May [online] https://web.archive.org/web/20151106011516/https://www.idmanagement.gov/sites/default/files/documents/SHA256_Transition_Lessons_Learned.pdf (accessed 14 August 2019).

SPHINCS+ Team (2018) *Reference Implementation* [online] https://sphincs.org/software.html (accessed 14 August 2019).

Truskovsky, A., Lafrance, P., Van Geest, D., Fluhrer, S., Kampanakis, P., Ounsworth, M. and Mister, S. (2018) *Multiple Public-Key Algorithm X.509 Certificates*, Internet-Draft draft-truskovsky-lamps-pq-hybrid-x509-00, IETF Secretariat, March [online] http://www.ietf.org/internet-drafts/draft-truskovsky-lamps-pq-hybrid-x509-00.txt (accessed 14 August 2019).

Notes

1 Strictly, the mechanism allows placing any second public key and signature in the extensions; it is not restricted to PQ cryptography.

2 More details are available at the authors' website https://sphincs.org/ (access 5 August 2019).

3 The version of OpenSSL 1.0.2k-fips 26 January 2017 we used is within CentOS 7 and might have been modified or fixed to fit the operating system, while the version 1.1.1b 26 February 2019 was directly downloaded from the website of the OpenSSL project and installed on CentOS 7.

4 The API could be called right after, for example, https://github.com/openssl/openssl/blob/master/apps/s_client.c\#L1734 (access 5 August 2019).

5 CFSSL is an open source PKI/TLS toolkit, available at https://github.com/cloudflare/cfssl (access 5 August 2019).

6 The API could be called right after, for example, https://github.com/openssl/openssl/blob/master/apps/ocsp.c\#L1555 (access 5 August 2019).

7 Wireshark is a widely-used network protocol analyser, available at https://www.wireshark.org/\#download (access 5 August 2019).

8 The manpage is available at https://www.openssl.org/docs/man1.0.2/man1/openssl-ocsp.html (access 5 August 2019).