The Viability of Post-Quantum X.509 Certificates

Panos Kampanakis, Peter Panburana, Ellie Daw[1] and Daniel Van Geest[2]

[1] Cisco Systems, USA,
{panosk, pepanbur, edaw}@cisco.com,
[2] ISARA, Canada,
Daniel.VanGeest@isara.com

Abstract. If quantum computers were built, they would pose concerns for public key cryptography as we know it. Among other cryptographic techniques, they would jeopardize the use of PKI X.509 certificates (RSA, ECDSA) used today for authentication. To overcome the concern, new quantum secure signature schemes have been proposed in the literature. Most of these schemes have significantly larger public key and signature sizes than the ones used today. Even though post-quantum signatures could work well for some usecases like software signing, there are concerns about the effect their size and processing cost would have on technologies using X.509 certificates. In this work, we investigate the viability of post-quantum signatures in X.509 certificates and protocols that use them (e.g. TLS, IKEv2). We prove that, in spite of common concerns, they could work in today's protocols and could be a viable solution to the emergence of quantum computing. We also quantify the overhead they introduce in protocol connection establishment and show that even though it is significant, it is not detrimental. Finally, we formalize the areas of further testing necessary to conclusively establish that the signature schemes standardized in NIST's PQ Project can work with X.509 certs in a post-quantum Internet.

1 Introduction

X.509 [1] defines public key certificates that are used to authenticate entities via signatures from publicly trusted authorities. These certificates are used in IETF's Public Key Infrastructure (PKI) X.509 (PKIX) standards and are also widely deployed on the Internet for authentication. X.509 certificates are PEM or DER encoded. Their size can vary based on the attributes, algorithms (ECC, RSA) and keys in them. The most common certificate sizes on the Internet today vary between 500-1500B. In PKI, verifiers have a pre-populated truststore of root certificates they already trust. In order to verify a certificate that contains an entity's public key, the verifier needs to be able to authenticate a certificate chain (sequence of signatures) from the leaf to be authenticated all the way to the root certificate, in its pre-trusted truststore. These certificate chains used on the Internet today usually consist of two to four certificates with a total size of a few kilobytes (KB). Thus in today's world, X.509 certificates and their corresponding certificate chains add up to a few kilobytes of data that are exchanged by the entities authenticating.

In a post-quantum (PQ) world, the public key algorithms used to generate a certificate's public key, which is then signed by the certificate authority's private key, are susceptible to being broken. A quantum computer (using Shor's algorithm [2, 3]) could derive the private key corresponding to the public key in a certificate; this is significant because it means that an attacker could impersonate the certificate owner at any time. In order to address this concern, many post-quantum signature schemes have been proposed in the literature [4–13].

Most of the proposed PQ schemes have public key and signature sizes of 10-200KB [14], which is considerably bigger than today's typical 500B, or less, key and signature sizes. Table 1 shows roughly the size of the public key and signature in some classes of traditional and quantum-resistant signature algorithms. It shows that the sizes of the signatures and keys of quantum-resistant schemes can grow from a few to many kilobytes which can pose challenges for today's existing infrastructures that would use them in X.509 certificates. These challenges include transmission overhead which leads to delays in the connection completion, IP fragmentation due to their size being bigger than the Path Maximum Transfer Unit (MTU) and wasted bandwidth for connections that transfer small amounts of data. Similarly, NIST Round 1 [21] PQ signature candidates vary from 2KB to a few MB.

	PK+Sig. size (KB)
ECDSA	~ 0.1
RSA	~ 0.5
Stateful HBS	15
Stateless HBS	42
Code-based	190
Lattice-based	11
Multivariate	99
SS Isogenies	122

Table 1: Approximate traditional and post-quantum public key and signature sizes

To support the algorithms used in the X.509 or other certificate standards [15, 16], some straightforward changes will be needed. RFC5280 and RFC6818 define the X.509 PKI certificate and CRL profile used on the Internet. These standards describe an Algorithm Identifier used in the X.509 certificate structures. Algorithm Identifiers used in X.509 certificates are defined in various other standards, such as RFC2528 which was obsoleted by RFC3279 and further updated by RFC4055, RFC4491, RFC5480, RFC5756 and RFC5758. These X.509 algorithm identifiers are also used in multiple other standardized protocols like EST (RFC7030) and PKCS#10 (RFC2986 updated by RFC5967). Thus, to use new post-quantum signatures in X.509, changes would be required in the X.509 algorithms. In a similar example, [17] defined the corresponding identifiers to support EdDSA signatures. Thus, it is straightforward to add support for new proposed post-quantum signature schemes in X.509 when necessary by defining new algorithm identifiers (that correspond to certain post-quantum signature scheme parameters and structures). Alternatively, hybrid PQ certificates [18,19] that are backwards compatible with traditional X.509 can be defined to be used while post-quantum and pre-quantum compatibility is required. These certificates make use of non-critical extensions and PQ algorithm identifiers to add a PQ public key and signature on top of the traditional key and signature in X.509.

Our contribution: Due to the size challenges that PQ signatures schemes introduce, in this work we evaluate how practical such signatures are in protocols that use X.509 today. We analyze the fragmentation mechanisms in these protocols and experimentally evaluate our analysis. We conjecture that the lengthier post-quantum certificates will still be usable in existing protocols. We do not consider the signature processing overhead which needs to be studied on a per algorithm basis [20], but we focus on the protocol transmission overhead introduced by the certificate size. Signature generation and verification of most PQ schemes proposed in [21] are more processing intensive than RSA or ECDSA, but specific PQ signature scheme processing overhead will be studied separately. Finally, we formalize the areas of further testing necessary to conclusively establish that the schemes standardized in NIST's PQ Project will work with X.509 certs in a post-quantum Internet.

The rest of the paper is organized as follows: Section 2 describes the implications introduced by big PQ certificates to widely used protocols and how they are addressed by built-in protocol mechanisms. Section 3 validates our analysis experimentally. Section 4 summarizes the findings, discusses usecases where PQ certificates would not be viable and establishes future work.

2 PQ certificate Implications

2.1 (D)TLS

(D)TLS is a ubiquitous protocol used to protect communications (i.e., browsers, EAP-TLS, SSL VPN). (D)TLS authentication often depends on public key X.509 certificates. A certificate is usually used to authenticate the identity of a server or client. By leveraging a pre-trusted set of root certificate authorities, the client can use the certificate chain and the server's certificate provided in the (D)TLS handshake to verify that this is a certificate it can trust and that the server's identity is what is expected.

In a post-quantum world, due to their size, post-quantum public keys and signatures will significantly increase the size of the certificate chains being exchanged with (D)TLS records in the (D)TLS handshake, and therefore have the potential to introduce substantial overhead to the exchanges. The protocols already have mechanisms in place to address this concern:

TCP Segmentation / (D)TLS Record Fragmentation The TLS protocol supports the ability for a certificate or certificate chain to exceed the maximum payload length that is supported in the network path between the client and the server. For TLS records that do not exceed 16KB, TLS uses TCP segmentation in order to split apart long data payloads. In this case, the data which is known to be too long is split into smaller pieces before being sent to the IP layer to be packaged and sent. The receiver is then responsible for reassembling the segments. Given that the MTU is commonly 1500B in modern networks, even today, certificate chains often require segmentation in TLS.

In the event of a post-quantum certificate or chain exceeding the 16KB maximum length, TLS leverages Record Fragmentation, a mechanism that allows for the sender to fragment his lengthy TLS records before sending them [22, 23]. In this case, the packets are split after being packaged and recognized as too large. The receiver coalesces the records that, when pieced together, contain the original record. Similarly for DTLS [24, 25], it allows for fragmentation of the handshake messages over multiple records. In both of these cases, the certificate introduces overhead at the protocol level in the form of additional packets being sent across the wire. However, the number of packets will not affect the completion of (D)TLS handshakes. Therefore, large certificate chains are likely to introduce record fragmentation which will lead to an increased number of record exchanges and delay, but the protocols will still be able to complete the handshakes successfully. It is important to note that there is a potential for additional overhead in the form of authenticating (signature generation and verification) these large-sized certificates and public keys, which highly depends on the signature algorithm. **Note**: Other than certificate authentication, (D)TLS record fragmentation will also benefit post-quantum key exchange messages with larger keys.

Per Connection Overhead It is obvious that heavy post-quantum (D)TLS handshakes would have significant impact on connections that transfer small amounts of data. For example, transferring 20KB of certificates before sending only 1KB of data introduces relatively massive overhead to the connection. In today's Internet, HTTP/1.1 uses multiple connections in order to serve resources to the client. Leveraging multiple costly established connections with large post-quantum certificate chains in order to transfer small amounts of data would negatively impact HTTP/1.1 web pages.

On the other hand, HTTP/2 [26] introduces several improvements by multiplexing data over one single connection. Multiplexing alleviates the overhead introduced by the heavy handshake by amortizing the overhead across a greater number of total bytes being transferred over the same connection. HTTP/2 adoption on the Internet has doubled to 14.6% over the last year [27]. At of the time of this writing, the average number of requests per web page was about 100, the total transferred data was about 2500KB and the average number of connections per page was about 33 [28]. It is evident that with the adoption of multiplexing in HTTP/2, the overhead introduced by heavier TLS handshakes will be better amortized across more data heavy connections. 1-2% overhead for a multiplexed TLS connection that transfers a total of 2MB of data over 100 requests is acceptable. Although the average case will not suffer significantly, there will be pages where the resources pulled by various servers are so small that an PQ TLS handshake will add significant overhead. For example, the performance of pulling 33 different 1KB resources would suffer if we introduced a heavy PQ TLS handshake. Client caching would be the mechanism that alleviates the overhead in subsequent connections in this case.

Caching RFC7924 defines how information exchanged in the (D)TLS handshake could be significantly decreased by the client indicating to the server what certificate it has cached for him already from a previous handshake. The server can avoid resending a certificate or chain if it is already cached by the client. When the highlighted concern for protocol overhead in a post-quantum world is the sending of large size certificate chains, this offers an obvious advantage. Even though caching offers apparent advantages, it also has certain shortcomings. Since it uses an old session to authenticate a new one, it leaks information in the unencrypted (D)TLS handshake about the server certificate which could allow for the correlation of (D)TLS sessions. For TLS 1.3, an attacker that observed a ClientHello could replay it using its own KeyShare and read the resulting encrypted certificate reply. The Security Considerations section of RFC7924 discusses some of these concerns. Even though potentially beneficial for reducing the protocol overhead of bigger certificates, caching's security concerns prevent us from considering it as a general solution to the challenges big PQ certificates pose.

Compression Certificate compression is a mechanism that shrinks the size of the certificates transferred between communicating parties. Compression comes with disadvantages which have been discussed in IETF. Since compression adds processing for the client, it introduces Denial of Service (DoS) concerns. Although we are not considering the processing overhead of PQ-sized certificates, it is important to acknowledge that the cost of compression and decompression processing should be less than the cost of protocol transmission of the decompressed messages. Specifically for X.509, some preliminary testing showed us that compressing a regular DER encoded RSA certificate gives only a 20% size improvement for the certificate, which comes from the compression of recurring text seen in the certificate. It is clear that reducing the certificate size by 20% may not introduce considerable improvements in the overhead during certificate transmission. In a PQ certificate that consists of 90% randomly generated key and signature structure text, the compression benefit would not be significant (much less than 20%), and therefore the improvements in transmission overhead would be negligible. Thus, even though IETF is working on a new compression draft [29], the benefits that compression can offer for big post-quantum certificates do not seem to outweigh the challenges it introduces.

2.2 QUIC

QUIC [30] is a transport protocol that runs over UDP. It was designed to provide better performance and congestion control than HTTP over TLS using TCP for the web. Similar to DTLS, QUIC mandates fragmenting big payloads so they can fit within the specified MTU in the path of the communicating parties. Thus, large PQ certificates included in the QUIC messages would be fragmented by the sender, and later reassembled by the receiver. Additionally, as with HTTP/2, QUIC leverages multiplexing of multiple streams which will improve the amortization of big certificate exchanges in a new QUIC connection. QUIC also uses gzip compression for certificates and supports caching of certificate chains. Multiplexing, fragmentation, compression, and caching are mechanisms which alleviate the challenges introduced at the transport layer by post-quantum signatures. We expect that the QUIC protocol will not be detrimentally impacted by PQ certificates, which is emulated in Section 3.2

2.3 IKEv2

IKE is a protocol that establishes VPN keys to protect communications. IKEv2 [31] uses X.509 certificates to authenticate peers that are establishing a VPN tunnel on the Internet. It almost always runs over UDP. As with DTLS, large PQ certificates carried in IKEv2 messages over UDP introduce fragmentation concerns. To address these concerns, RFC7296 defines the use of a hash and an URL that serve the certificate in order to avoid sending it over UDP. This method is almost never used in IKEv2 deployments today. Another more commonly used option is described in RFC7383 [32] that defines fragmentation for IKE_AUTH messages which transport peer certificates in IKEv2. This option doesn't attempt to retransmit individual fragments, so if one fragment of the IKE_AUTH message is lost in transit the entire IKE_AUTH message needs to be retransmitted which would introduce IKEv2 negotiation delays in lossy networks. RFC7383 is supported in the VPN stack of various vendors like Apple and StrongSwan. Networking vendors like Cisco and Juniper also support IKEv2 fragmentation.

Today, IKEv2 fragmentation is used for traditional certificate chains that exceed the size of the path MTU. Large post-quantum certificates will be transferred over multiple IKEv2 fragments with some delay but without special issues as well. IKEv2 authentication takes place only during the initial IKEv2 IKE_AUTH exchanges. Subsequently, no further authentication messages are exchanged after the VPN tunnel is established unless RFC4478 [33] is used. Thus, there will be good amortization of the fragmented big PQ certificates over the encrypted data transferred over the lifetime of a VPN tunnel. **Note**: IKEv2 fragmentation will not benefit post-quantum key exchange as it applies only to IKE_AUTH and not IKE_SA_INIT that performs a public key exchange.

3 Experimental Results

From our analysis it is evident that PQ certificates would introduce some overhead in protocols that use X.509 today, but the mechanisms already presented in these protocols alleviate the concerns stemming from the transmission of large-sized certificates. Below we try to experimentally prove our analysis by using hash-based signature certificates and implementing them in TLS and IKE software libraries. We also test (D)TLS, QUIC and IKE by emulating post-quantum certificates with long, big-sized certificate chains.

3.1 Using large Hybrid X.509 PQ HSS certificates

We first implement a hybrid X.509 scheme that combines traditional and post-quantum signatures and public keys [18, 19]. We begin as if we are creating a standard X.509 certificate. Included in this initial certificate are two additional non-critical extensions, one specifying the subject's post-quantum public key and the other specifying the post-quantum algorithm with which the issuer will create a signature. These extensions serve similar purposes in the PQ world as the SubjectPublicKeyInfo and SignatureAlgorithm fields of a traditional X.509 certificate. The certificate is then encoded and the encoding is signed by the issuer's post-quantum private key using the specified algorithm. This signature is then added to the certificate in a third non-critical extension. This extension serves a similar purpose in the PQ world as the Signature field of a traditional X.509 certificate. The hybrid certificate containing all three extensions is then signed as usual with the issuer's traditional RSA or ECDSA key. When the certificate is constructed in this way, a verifier who doesn't understand the new extensions can still verify the traditional signature, while a verifier who understands the extensions can verify the post-quantum signature. An example hybrid certificate is shown below. With minor modifications, protocols which sign challenges or portions of their transactions with the traditional private key can negotiate to sign with the post-quantum private key correspond to the post-quantum public key.

```
X.509 Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 4097 (0x1001)
    Signature Algorithm: ecdsa-with-SHA256
        Issuer: C=US, ST=NC, O=CISRA, CN=HSS-Hybrid-CACert-Test
        Validity
            Not Before: Jan  9 17:33:02 2018 GMT
            Not After : Jan  9 17:33:02 2019 GMT
        Subject: C=US, ST=NC, O=CISRA, CN=HSS-Hybrid-ServerCert-Test
        Subject Public Key Info:
            Public Key Algorithm: id-ecPublicKey
                Public-Key: (256 bit)
                [ ... omitted for brevity ... ]
        X509v3 extensions:
            X509v3 Basic Constraints:
                CA:FALSE
            Netscape Cert Type:
                SSL Server
            Netscape Comment:
                OpenSSL Generated Server Certificate
            [ ... omitted for brevity ... ]

            Alt-Signature-Algorithm:
                sha512WithHSS

            Subject-Alt-Public-Key-Info:
                Leighton-Micali Hierarchical Signature System
                Public Key:
                    00:00:00:01:00:00:00:07:00:00:00:03:1c:ba:ef:
                    [ ... omitted for brevity ... ]
                Winternitz Value:   3 (0x3)
                Tree Height:   7 (0x7)

            Alt-Signature-Value:
                Signature:
                    30:82:0a:74:[ ... omitted for brevity ... ]

     Signature Algorithm: ecdsa-with-SHA256
          30:45:02:21:[ ... omitted for brevity ... ]
```

We implemented hybrid X.509 certificates in OpenSSL and StrongSwan, two open-source libraries, in order to test TLS 1.2 and IKEv2 respectively. Both libraries (OpenSSL 1.0.1l, StrongSwan 5.5.0) were updated to be able to parse and verify these certificates by using the traditional and PQ signatures and sign messages with the corresponding public key in order to perform authentication in TLS and IKEv2. The post-quantum signature algorithm we used in the hybrid certs was hash-based HSS [6] with various tree size and Winternitz chain parameters. The host we tested in was running Ubuntu 16.04 on an Ivy Bridge i3. The handshake times we measured for TLS were from the initial ClientHello to the Server Hello Done. The systems testing the algorithms were three network hops away from each other.

Our experiments share some common ground with the experiments in [19] that emulates hybrid PQ certificates in TLS libraries and browsers, but does not test real PQ algorithms like HSS. Instead, it uses big-size extensions which are practically not evaluated during signature generation in hybrid certificates and do not add processing overhead. Our experiments use real PQ hash-based signatures which are more costly to generate and verify.

	RSA	HSS ($w = 4$, $h = 20$, $L = 1$)	HSS ($w = 8$, $h = 25$, $L = 1$)	HSS ($w = 8$, $h = 20$, $L = 1$)
Handshake time	0.073s	0.343s	0.392s	0.381s
Avg # Packets	3	7	5	5

Table 2: TLS handshake with Hybrid HSS cert chains of length= 1 vs traditional 2Kb RSA key cert chains of length= 2. Client-server 3 hops away.

The results are summarized in Tables 2 and 3. For both TLS and IKEv2 experiments, the hybrid certificate chain was of length one. The TLS packets counts were the number of certificate chain segments sent from the server. To compare with traditional RSA certs we used a popular bank website a few hops away that uses a 2048-bit cert chain of length two. As we can see in Table 2, in TLS we saw, as expected, handshake times and total number of packets transmitted increase from our baseline of seven handshake packets and 0.073s. Focusing on the number of packets, we saw in each scenario modest increases in the number of handshake packets due to increase in size of the certificates transferred. Depending on the size of the public key and signature of the PQ scheme, the cert chain size could significantly change, but TLS record segmentation should allow for the successful transfer of these chains. As for the handshake times, even though it is specific to the signature algorithm and the network topology (hops between client and server), Table 2 includes the signature generation and verification times for HSS along with the transmission delay due to the extra fragments. For the HSS parameters chosen, signature generation and verification were not very expensive. Adding 200-300ms to a handshake is negligible for most usecases. Depending on the PQ signature scheme the processing of these signatures could vary.

For IKEv2, we measured the IKE negotiation times from the first IKE_SA_INIT to the last IKE_AUTH message and the number of IKE_AUTH fragments coming each direction using PQ HSS certificates. The systems testing the algorithms were locally connected. As we can see in Table 3, similarly to TLS, the negotiation took 200-400ms more time and the IKE_AUTH fragments increased. For both HSS parameter sets, we observed fragmentation during session establishment. When using a RSA based certificate with a 2048-bit key, no fragmentation was observed and an SA was established in 0.09s with four packets. All IKEv2 sessions were able to be established in under half a second with HSS based certificates. These experiments prove that IKEv2 has the transport mechanisms needed to handle the increased sizes of HSS certificates, or other PQ schemes with big keys and certs. Additionally, the delay introduced is not detrimental, but it could vary based on signature algorithm used and network topology (hops between client and server).

In summary, we showed that the size of HSS post-quantum signatures used in common protocols like TLS and IKEv2 will not deem them unusable. As explained in our analysis, both protocols contain fragmentation mechanisms that

	RSA	HSS ($w = 4$, $h = 20$, $L = 1$)	HSS ($w = 8$, $h = 15$, $L = 1$)
Negotiation time	0.09s	0.40s	0.26s
Avg # Packets	1	5	3

Table 3: IKEv2 negotiation with Hybrid HSS cert chains of length= 1 vs traditional 2Kb RSA key cert chains length= 1. Client-server 3 hops away.

allow for fragmenting big TLS records and IKE_AUTH messages without the extra packets and transmission delay being an important factor for the completion of the negotiation. In our experiments we did not focus on the signature generation and verification times required for the TLS and IKE handshakes. These could vary based on the PQ algorithm chosen, but as a proof-of-concept we tested the total handshake times that included signing and verification with HSS. HSS ad reasonable time performance for the chosen parameters.

3.2 Emulating large PQ X.509 certificates

To further quantify our analysis we tested the aforementioned protocols with large certificates of comparable size to the PQ certificate sizes. Even without using actual post-quantum algorithms to generate the certificates, our experiments reproduce the transmission overhead that would be introduced by testing with certificates of similar size. In the experiments below we focus on the transmission overhead as we want to prove that existing protocol mechanisms would ensure the big PQ certificates are exchanged with no issues. We are not addressing the potential delay of signature generation and verification which would depend on the exact PQ signature scheme used and the topology. We also are not interested in shortcomings of protocols when parsing long certificate chains as long as these chains were transmitted successfully.

Our experiments share some common ground with the experiments in [19] that emulates hybrid PQ certificates in TLS libraries and browsers, but does not focus on emulating PQ-sized certs in QUIC or IKEv2. Another distinction is that in our tests we use long certificate chains that add up to a significant size and increases the chain processing overhead (as expected in with a more costly PQ algorithm) of signature verification. [19] uses big-size extensions in hybrid certificates which are practically not evaluated during signature generation and do not add processing overhead. Even though we do not prove the processing overhead of a lengthy chain in our test is the same as with a PQ signature scheme, we believe that our emulation can be considered closer to reality.

(D)TLS with OpenSSL - WolfSSL To test (D)TLS 1.2, we used OpenSSL 1.0.2g, WolfSSL 3.11.1, and Apache Web Server 2.4.25. Like OpenSSL, WolfSSL is a library that implement (D)TLS, and the Apache Web Server is commonly used in Internet web servers. X.509 certificates were used for server authentication. To emulate today's certificate sizes we employed 2048-bit RSA public keys, which are very common in today's PKI deployments. ECC keys used in digital certificates are of much smaller size. To simulate large PQ certificates we used 8192 and 16384-bit RSA public keys in certificates with multiple intermediate CAs. The certificate chains amounted to 8, 18 and 19KB for the above key sizes respectively. To extensively test applications with even bigger chains we also created 21 and 24 certificate long chains with 1024 and 16384-bit keys and multiple SAN attributes that amounted to 23 and 135KB respectively.

Table 4a shows the results from testing OpenSSL client against Apache server. We could verify that the post-quantum size certificate chains were being transferred in the TLS 1.2 handshake by using segmentation and record fragmentation. As shown in the table, 24 cert long, 135KB chains with 16Kb keys were failing at the OpenSSL client. Packet captures confirmed that the TLS records were successfully received, but the client was introducing the error due to message size limits. The 21 long cert chain of total size 23KB was transferred and processed by the client with no failures. Our results are futher reinforced in [19] that tested TLS 1.2 hybrid PQ certificates with other libraries like GnuTLS, mbedTLS, BouncyCastle and NSS.

Similarly for DTLS, we used WolfSSL with the WolfSSL DTLS example client/server. As shown in 4c, DTLS worked correctly with lengthy server certificates (with 8, 16Kb RSA keys) that required fragmentation. After receiving the big server certificate, the WolfSSL client was reporting an error because of a buffer size limit, but packet captures confirmed that DTLS fragmentation was working correctly and the client was receiving the certificate successfully.

To further examine the correct operation of TCP segmentation separately, we tested various public web servers by using a very small interface MTU size which caused TCP segmentation of the certificate chains exchanged in the TLS handshake. Our experiments showed that, as expected, TCP segmentation will operate fine for big post-quantum certificates and chains that do not exceed the 16KB TLS record limit.

We then investigated the transmission overhead introduced by the post-quantum certificates. We compared the average handshake time of a TLS handshake with 16Kb public keys and a chain of 19KB to the handshakes with 2Kb keys and 8KB chains. The total time increase was almost 0.5 seconds for a client-server distance of three, which is negligible for most TLS connection requirements. The corresponding number of packets from the server required to complete the handshake increased by 70%. Even though there are significant increases in the handshake time and packets exchanged, they are acceptable for most usecases. Similarly for DTLS, the DTLS handshake took longer and required more packets to complete. For the 16Kb key certificate, the WolfSSL application was failing due to key size limits, which prevented us from collecting accurate statistics of the increases in DTLS, but the overall increases were similar to the experiments with TLS.

TLS with Browsers In order to study the web TLS implementations we tested the Chrome and Firefox browsers. The versions we worked with were Chrome

	OpenSSL	Firefox	Chrome
8-20KB cert chain, (2, 8, 16)Kb pub. keys	✔	✔	✔
24 long cert chain, 1Kb pub. keys	✔	✘*	✘*
21 long cert chain, 16Kb pub. keys	✘*	✘*	✘*

(a) TLS

	proto-quic
24 long cert chain, 1Kb pub. keys	✔
21 long cert chain, 16Kb pub. keys	✘*
8KB cert chain, 2Kb pub. keys	✔
19KB cert chain, 16Kb pub. keys	✔

(b) QUIC

	WolfSSL
2Kb pub. key RSA server cert	✔
(8, 16)Kb pub. key RSA server cert	✘*

(c) DTLS

	Cisco	StrongSwan
8KB cert chain, 2Kb pub. keys	✔	✔

(d) IKEv2 (IKE_AUTH fragmentation)

* Failures related to the application certificate chain parsing, not to record fragmentation or segmentation.

Table 4: Experimental results of (D)TLS, IKEv2, QUIC using big X.509 certificates

58.0.3029.110 (64-bit) that uses the BoringSSL library and Firefox 53.0.2 (Windows 32-bit and Ubuntu 64-bit) that uses the NSS library. We were able to verify that TCP segmentation and TLS record fragmentation were working correctly. When using a certificate chain of total size 19KB with 16Kb public keys and 18KB chain with 8Kb key certificates respectively, the TLS handshake was completing successfully. On the other hand, when using a certificate chain of 135KB with 16Kb keys, the handshake was failing. By using packet captures, we were able to see the certificates transferred correctly in the handshakes which meant that the failures were introduced by the browsers themselves when parsing the chains, not by the protocols. When using a 1Kb RSA public key certificate with a chain of length 24 we were able to see the browsers parsing part of the chain before failing. Thus, it was evident that regardless of the size of the chain, the chain length was the trigger of these failures in the browsers, and not the chain size itself. The conclusion from these experiments was that PQ size certificates will work slightly slower, but as expected in (D)TLS connections. Our results are further reinforced in [19] that tested TLS 1.2 hybrid PQ certificates with other browsers like Apple Safari, Edge and Opera.

QUIC uses fragmentation, compression, and caching to minimize the impact of certificates in the handshake. In order to test how QUIC would operate with quantum-resistant X.509 certificates we used Google's proto-quic implementation. We tested 23KB certificate chains, 24 long with 1Kb RSA keys, 8KB chains with 2Kb keys and 19KB chains with 16Kb keys (Table 4b) in order to emulate similar to post-quantum cert sizes. We were able to verify that all chains were transferred correctly between the client and the server by using fragmentation and compression. Due to the vast majority of the text within a certificate consisting of non-compressible random RSA signature values the benefit of compression was less than 20%. We also tested 21 long certificate chains with 16Kb keys which failed in the proto-quic client. The error was not because of the protocol dealing with the certificate sizes since we were able to verify that the fragments were transmitted successfully. Instead, the error was due to a few second delay being introduced by the server while transmitting the certificate chain to the client, which led to a client timeout. Thus, the failure was triggered by the implementation and not the protocol itself. It is evident that QUIC will operate with lengthy PQ certificates with no issues.

IKEv2 is widely deployed today. It is not rare that large X.509 certificates and certificate chains need to be transferred over UDP as part of the IKEv2 peer authentication. Thus, fragmentation is already widely used on the Internet today in order to carry lengthy certificates that do not fit in the path MTU. In order to verify that IKEv2 fragmentation defined in RFC7383 will also work with PQ size certificates, we tested StrongSwan 5.5 and Cisco IOS-XE software with 2Kb public keys in an 8KB, 5 long certificate chain. We were able to confirm that, as expected, IKE_AUTH fragmentation of these long certificates was working correctly (Table 4d).

4 Conclusion and Future Work

In conclusion, in this work we wanted to analyze the impact of large PQ signatures in X.509. We studied the transmission overhead introduced by large certificates in various protocols. We looked into TLS, DTLS, IKEv2 and QUIC that use digital certificates for authentication. We evaluated existing mechanisms built-in the protocols that deal with large records like record fragmentation, segmentation, caching, and compression. We proved that mechanisms like segmentation and fragmentation would add some more overhead by breaking the large PQ certificate chain in more fragments, but still allow for large certificates to be transferred successfully. The overhead is not negligible, but can be acceptable in most modern usecases. We also showed that hash-based HSS post-quantum hybrid certificates could operate in these protocols.

Newly introduced methods like compression and caching in the QUIC protocol will also help to alleviate the overhead of large X.509 certificates, but introduce some security concerns and thus are not considered preferable solutions to the large PQ cert problem. Multiplexing, which is already employed in HTTP/2 and QUIC, will further minimize the extra delays in (D)TLS handshakes attributed to these certificates by using one connection to transfer multiple data transactions. We also showed that even though the implementation of each protocol can have bugs and limitations on key and buffer sizes, the protocols alone can transfer the long PQ certificates chain with no issues, proving that the overhead in the protocol is not detrimental to the functionality. Implementation bugs will need to be addressed to eliminate oversize keys, signatures and buffer errors.

There will still be environments that are very sensitive to delays, cannot store big certificates, or perform PQ signature generation or verification because of their processing constraints. Such environments usually contain resource constrained devices or delay-sensitive communications. In such environments, most post-quantum signatures would be impractical because of their processing intensity. Other signatures like [34] could be alternative for these usecases.

Future Work: In spite of the arguments made in this work, we still need to expand our experiments to ensure a PQ PKI is possible without major changes in the existing protocols and infrastructure. In our TLS experiments, we verified a root CA signature, the leaf certificate signature and the TLS/IKEv2 handshake signature to verify the identity of the server. Real-world scenarios include signed certificate timestamp (SCT) and potentially an OCSP staple verification. Additionally, subCA signatures could also exist in the cert chain. Thus, the experiments shown in this work could include half the amount of PQ signature verifications of a real-world scenario. The additional handshake data and processing could double, which would not be detrimental based on Table 2. Even though we proved that larger signature data will be transferred using fragmentation, their impact on the overall handshake completion time would need to be more carefully evaluated especially for real-world Internet paths over multiple network hops.

Moreover, in this work, we tested with stateless hash-based signatures and hybrid certificates. Stateful signatures are possible, but require careful state management [35] in order to prevent forgery. Stateful signatures can be more suitable for off-line rootCAs as state management would not be a concern for these scenarios. A stateless HBS scheme like SPHINCS+ [21] eliminates the state management need at the cost of bigger signatures and processing overhead. We intend to evaluate the practicality of using SPHINCS+ in PQ hybrid certificates.

Specifically for HBS, the total number of messages signed are important for the size of the tree. For example, a height 50 tree, would take more than 21 years to run out even if we signed 2^{20} messages per second. The total SCT, OCSP, subCA and root CA signatures are not expected to be as many as the TLS/IKEv2 handshake signatures. 2^{40} signed leaf certificates could be more than enough for a subCA and 2^{16} signed subCA certs could be enough for a root CA. Thus, smaller tree parameters for root CA, subCAs, OCSP and SCT signatures could alleviate the certificate chain size and processing cost. We intend to quantify these parameters to evaluate how small the cert chain size could become with HBS.

Additionally, HBS are not the only option. More lightweight algorithms (i.e. Dilithium, WalnutDSA) proposed in [21] need to be evaluated in order to confirm their practicality.

One more topic that needs further investigation is time-to-first byte for webpages on the Internet. If a website pulls resources from 33 different hosts, then performing 33 heavyweight PQ TLS handshakes could affect the time the page starts rendering in the browser. The first byte rendered could come from the first connection that starts transferring back data after any HTTP redirects, so it doesn't mean all handshakes and resources would need to have been downloaded before the page starts rendering. Non-cached resources would certainly take more time to complete downloading over PQ TLS handshakes, but the slowness might not be noticeable as long as the handshake that transfers the bulk of the page over HTTP/2 completes in sufficient time. More testing would need to be performed with real webpages in order to justify this intuition.

Recently, while testing TLS 1.3, the IETF found that middleboxes deployed on the Internet were interfering with TLS 1.3 handshakes they did not understand. Even though the hybrid certificates and TLS changes introduced in this work are not expected to "confuse" such middleboxes as the TLS 1.3 messages and data structures, the effect of large number of fragments in different TLS handshakes going over one TCP connection in HTTP/2 should be evaluated. Such fragmentation is already happening today with cert chains of length 3-4 and RSA keys of 2048 or 3072 bits, but we intend to experiment with such scenarios and PQ certificates to ensure that there will be no outages introduced by middleboxes on the Internet.

Acknowledgments

The authors would like to thank Graham Bartlett for his valuable help with fragmentation in IKEv2. We would also like to thank Alex Truskovsky for his help with HSS. Hanno Böck also was kind enough to challenge us very insightfully on the concerns PQ signatures bring to PKI infrastructures.

References

1. International Telecommunications Union, "X.509: Information technology - open systems interconnection - the directory: Public-key and attribute certificate frameworks," https://www.itu.int/rec/T-REC-X.509/en.
2. P. W. Shor, "Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer," *SIAM J. on Computing*, vol. 26, no. 5, pp. 1484–1509, 1997.
3. J. Proos and C. Zalka, "Shor's discrete logarithm quantum algorithm for elliptic curves," *Quantum Info. Comput.*, vol. 3, no. 4, pp. 317–344, Jul. 2003. [Online]. Available: http://dl.acm.org/citation.cfm?id=2011528.2011531

4. J. Buchmann, E. Dahmen, and A. Hülsing, *XMSS - A Practical Forward Secure Signature Scheme Based on Minimal Security Assumptions.* Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 117–129. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-25405-5_8
5. A. Hülsing, D. Butin, S. Gazdag, and A. Mohaisen, "XMSS: Extended Hash-Based Signatures," https://datatracker.ietf.org/doc/draft-irtf-cfrg-xmss-hash-based-signatures/, 2016, Internet-Draft. Accessed 2016-06-06.
6. D. McGrew and M. Curcio, "Hash-Based Signatures," https://datatracker.ietf.org/doc/draft-mcgrew-hash-sigs/, 2017, Internet-Draft. Accessed 2016-06-06.
7. D. J. Bernstein, D. Hopwood, A. Hülsing, T. Lange, R. Niederhagen, L. Papachristodoulou, M. Schneider, P. Schwabe, and Z. Wilcox-O'Hearn, "SPHINCS: Practical Stateless Hash-Based Signatures," in *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques*, 2015, pp. 368–397.
8. D. J. Bernstein, T. Lange, and C. Peters, "Attacking and defending the mceliece cryptosystem," in *Proceedings of the 2Nd International Workshop on Post-Quantum Cryptography*, ser. PQCrypto '08. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 31–46. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-88403-3_3
9. N. Courtois, M. Finiasz, and N. Sendrier, "How to achieve a mceliece-based digital signature scheme," in *Proceedings of the 7th International Conference on the Theory and Application of Cryptology and Information Security: Advances in Cryptology*, ser. ASIACRYPT '01. London, UK, UK: Springer-Verlag, 2001, pp. 157–174.
10. P. S. L. M. Barreto, P. Longa, M. Naehrig, J. E. Ricardini, and G. Zanon, "Sharper ring-lwe signatures," *IACR Cryptology ePrint Archive*, vol. 2016, p. 1026, 2016.
11. L. Ducas, A. Durmus, T. Lepoint, and V. Lyubashevsky, *Lattice Signatures and Bimodal Gaussians.* Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 40–56. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-40041-4_3
12. J. Ding and D. Schmidt, *Rainbow, a New Multivariable Polynomial Signature Scheme.* Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 164–175. [Online]. Available: http://dx.doi.org/10.1007/11496137_12
13. A. Petzoldt, S. Bulygin, and J. Buchmann, *Selecting Parameters for the Rainbow Signature Scheme.* Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 218–240. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-12929-2_16
14. Y. Yoo, R. Azarderakhsh, A. Jalali, D. Jao, and V. Soukharev, "A post-quantum digital signature scheme based on supersingular isogenies," Cryptology ePrint Archive, Report 2017/186, 2017, http://eprint.iacr.org/2017/186.
15. *IEEE Std 1609.2-2016 (Revision of IEEE Std 1609.2-2013): IEEE Standard for Wireless Access in Vehicular Environments–Security Services for Applications and Management Messages.* IEEE, 2016. [Online]. Available: https://books.google.com/books?id=cwupAQAACAAJ
16. *ETSI TS 103097 - Intelligent Transport Systems (ITS) Security; Security hHeader and certificate formats.* ETSI, 2013. [Online]. Available: https://goo.gl/AgXkrb
17. S. Josefsson and J. Schaad, "Algorithm Identifiers for Ed25519, Ed448, X25519 and X448 for use in the Internet X.509 Public Key Infrastructure," Internet Engineering Task Force, Internet-Draft draft-ietf-curdle-pkix-04, Mar. 2017, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-ietf-curdle-pkix-04
18. M. Brown, "Quantum Safe PKI Transitions," ICMC presentation, 2017, https://icmconference.org/wp-content/uploads/G24a-Brown.pdf.

19. N. Bindel, U. Herath, M. McKague, and D. Stebila, “Transitioning to a quantum-resistant public key infrastructure,” in *Proc. 8th International Conference on Post-Quantum Cryptography (PQCrypto) 2017*, ser. LNCS, T. Lange and T. Takagi, Eds. Springer, June 2017, to appear.
20. S. F. Panos Kampanakis, “Lms vs xmss: A comparison of the stateful hash-based signature proposed standards,” Cryptology ePrint Archive, Report 2017/349, 2017, http://eprint.iacr.org/2017/349.
21. NIST, “Post-Quantum Cryptography Round 1 Submissions,” 2017, https://csrc.nist.gov/Projects/Post-Quantum-Cryptography/Round-1-Submissions.
22. T. Dierks and E. Rescorla, “The Transport Layer Security (TLS) Protocol Version 1.2,” RFC 5246, Aug. 2008. [Online]. Available: https://rfc-editor.org/rfc/rfc5246.txt
23. E. Rescorla, “The Transport Layer Security (TLS) Protocol Version 1.3,” Internet Engineering Task Force, Internet-Draft draft-ietf-tls-tls13-20, Apr. 2017, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-ietf-tls-tls13-20
24. E. Rescorla and N. Modadugu, “Datagram Transport Layer Security Version 1.2,” RFC 6347, Jan. 2012. [Online]. Available: https://rfc-editor.org/rfc/rfc6347.txt
25. E. Rescorla, H. Tschofenig, and N. Modadugu, “The Datagram Transport Layer Security (DTLS) Protocol Version 1.3,” Internet Engineering Task Force, Internet-Draft draft-ietf-tls-dtls13-00, Apr. 2017, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-ietf-tls-dtls13-00
26. M. Belshe, R. Peon, and M. Thomson, “Hypertext Transfer Protocol Version 2 (HTTP/2),” RFC 7540, May 2015. [Online]. Available: https://rfc-editor.org/rfc/rfc7540.txt
27. W3Techs, “Historical trends in the usage of site elements for websites,” https://goo.gl/fYmRyT.
28. http archive, “Trends,” http://httparchive.org/trends.php.
29. A. Ghedini and V. Vasiliev, “Transport Layer Security (TLS) Certificate Compression,” Internet Engineering Task Force, Internet-Draft draft-ietf-tls-certificate-compression-00, Jun. 2017, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-ietf-tls-certificate-compression-00
30. M. Thomson and J. Iyengar, “QUIC: A UDP-Based Multiplexed and Secure Transport,” Internet Engineering Task Force, Internet-Draft draft-ietf-quic-transport-03, May 2017, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-ietf-quic-transport-03
31. C. Kaufman, P. E. Hoffman, Y. Nir, P. Eronen, and T. Kivinen, “Internet Key Exchange Protocol Version 2 (IKEv2),” RFC 7296, Oct. 2014. [Online]. Available: https://rfc-editor.org/rfc/rfc7296.txt
32. V. Smyslov, “Internet Key Exchange Protocol Version 2 (IKEv2) Message Fragmentation,” Internet Requests for Comments, RFC Editor, RFC 7383, Nov. 2014. [Online]. Available: http://www.rfc-editor.org/rfc/rfc7383.txt
33. Y. Nir, “Repeated Authentication in Internet Key Exchange (IKEv2) Protocol,” Internet Requests for Comments, RFC Editor, RFC 4478, Apr. 2006. [Online]. Available: http://www.rfc-editor.org/rfc/rfc4478.txt
34. I. Anshel, D. Atkins, D. Goldfeld, and P. E. Gunnells, “Walnutdsa(tm): A quantum resistant group theoretic digital signature algorithm,” Cryptology ePrint Archive, Report 2017/058, 2017, http://eprint.iacr.org/2017/058.

35. D. McGrew, P. Kampanakis, S. Fluhrer, S. Gazdag, D. Butin, and J. Buchmann, "State Management for Hash-Based Signatures," in *Security Standardisation Research (SSR) 2016 - Lecture Notes in Computer Science*, vol. 10074. Springer, 2016.