Faster Post-Quantum TLS handshakes Without Intermediate CA Certificates

Panos Kampanakis[1] and Michael Kallitsis[2]

[1] AWS Cryptography, USA, kpanos@amazon.com
[2] Merit Network Inc., USA, mgkallit@merit.edu

Abstract. Traditionally, the most data-heavy part of a (D)TLS handshake has been authentication which includes a handshake signature and digital certificates. Although most common (D)TLS usecases are not significantly affected, some constrained ones such as low bandwidth environments or delay sensitive applications can see drastic performance degradation due to big certificates or certificate chains. That has led the security community to seek options to alleviate the issue. Post-quantum signatures and keys, on the other hand, have been proven to noticeably slow down handshakes even for common Internet (D)TLS or QUIC applications due to the significantly higher amounts of post-quantum authentication data they include. In this work, we quantify the size issue of post-quantum certificates in (D)TLS and QUIC and make the case for speeding up (D)TLS and QUIC handshakes by omitting the intermediate certificate authority certificates in the handshake. We present how that can be achieved along with the usecases that will mostly benefit from such a mechanism. We offer quantitative analyses to show that this approach is relatively straightforward, backwards compatible and with little overhead introduced for caching the certificates. We also discuss caching mechanisms based on different optimization goals.

Keywords: post-quantum TLS, PQ Authentication, post-quantum certificate chains, ICA suppression

1 Introduction

Digital communications have completely penetrated everyday life as enablers of numerous critical services including telemedicine, online banking, massive e-commerce, machine-to-machine automation, mobile and cloud computing. To ensure that it is secure, information is exchanged over secure tunnels which guarantee confidentiality and authenticity. Secure tunnel protocols (e.g. (D)TLS, QUIC, SSH) use cryptography to encrypt the data and Public Key Infrastructure (PKI) certificates to authenticate the communicating peers.

A PKI infrastructure consists of various parts. A Certificate Authority (CA) issues an entity's X.509 certificate [13] which assures the entity's identity and the public key (PK) tied to that identity. The identity is included in the Subject field of the certificate, while the entity's public key is stored in the Subject

Public Key Information along with the algorithm used by the issuer to create the signature. A certificate contains a specific validity period and extensions included by CAs to enable additional functionality. The certificate is signed by the CA's private key using the specified signature algorithm and the signature is added to the certificate's Signature field. The two most popular digital signature algorithms used in certificates today are the Elliptic Curve Digital Signature (ECDSA) and Rivest-Shamir-Adleman (RSA).

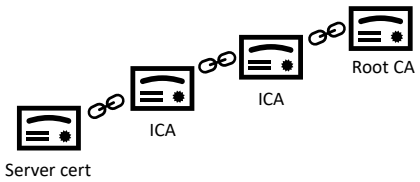


Fig. 1: Certificate Chain

At the top of the PKI, there are trusted CAs that self-sign their own certificates known as Root CA certificates. Normally a Root CA issues certificates for Intermediate CAs (ICAs). An ICA can further issue certificates for other ICAs that in turn sign leaf / entity certificates in the PKI. This process results in the creation of certificate chains of trust (Fig. 1) that usually consist of two to four certificates but can be arbitrarily long.

Widely used protocols like (D)TLS [7, 30, 32, 34], IKEv2/IPsec and QUIC [14, 48] leverage X.509 certificates and certificate chains rooted to a pre-trusted Root CA to authenticate a peer's identity and public key (PK). After exchanging ephemeral keys, the server (and optionally the client) sends its certificate chain to its peer along with a signature of the connection transcript which proves that it owns the identity private key corresponding to its identity certificate. The server's certificate is authenticated by verifying the certificate chain to a pre-trusted Root CA certificate. Given that certificate chains can be of arbitrary size, sometimes they could affect connection performance which we aim to alleviate in this work.

The **key contributions of our work** are summarized as follows:

(i) We quantify the heavy post-quantum authentication data issue in (D)TLS.

(ii) We quantify the amplification protection issue post-quantum algorithms introduce in QUIC.

(iii) We analyze public server datasets in order to quantify the number of ICAs used in (D)TLS connections today. We identify the ICAs and the types of certificates sent in the most popular servers' certificate chains. We analyze the numbers of distinct ICAs as the Top number of servers increases. We quantify the size of the ICA cache that someone would need to store ICAs in. Using these analyses we show that caching ICAs would speed up the connection handshakes without overloading most software applications used today.

(iv) We qualitative evaluate options of trimming down the authentication data in a handshake and propose ICA suppression, the most straightforward, backwards-compatible one.

(v) We propose ICA caching mechanisms which would allow entities to request ICA suppression.

(vi) We discuss considerations and security implications of ICA suppression and how they could be overcome.

The rest of the paper is organized as follows: Section 2 describes the issue with heavy authentication data in secure connection handshakes and presents usecases where it is more prevalent. Section 3 summarizes related work. Section 4 presents statistics of commonly used server datasets in order to quantify the number of ICAs an entity would need to store in order to implement ICA suppression. Section 5 describes options to alleviate the heavy authentication-data issue. Section 6 discusses the ICA caching mechanisms which could be used to alleviate (D)TLS handshakes. Section 7 goes over considerations and concerns with ICA suppression and potential protection mechanisms. Section 8 concludes this work and discusses future research topics.

2 The Heavy Authentication Data Issue

Although not the case for common Internet uses today, some important wireless sensor network protocols in the Internet-of-Things (IoT) ecosystem such as constrained mesh networks like Wi-SUN and IEEE 802.14.5 [23] usually operate in low-speed mediums. They often depend on EAP-TLS [43] for authentication. Using big certificate chains in such usecases could further exacerbate the low-speed issue and lead to authentications that fail or take minutes or hours [39, 21, 15].

What is more, recent developments in post-quantum cryptography are expected to lead to increases in the size of certificates which could slow down secure connection establishment even in typical Internet connections or applications. The cryptographic community has been researching quantum-secure algorithms for some time in order to address the quantum computer threat, and the US National Institute of Standards and Technology (NIST) has an ongoing, public project to standardize quantum-resistant public key algorithms. At the time of this writing, NIST's evaluation process is in Round 3 with 15 post-quantum (PQ) algorithm candidates remaining. Additionally, a few Internet Engineering Task Force (IETF) RFC drafts are already introducing these algorithms in IETF standards [28, 45, 49, 9, 22, 11, 10, 8]. When it comes to PQ signatures, the public key and signature sizes of NIST PQ signature candidates start from a few and can go up to tens of Kilobytes (KB) for the heavier schemes. The integration of such PQ signatures in X.509 certificates will naturally increase the size of these certificates significantly [17]. Certificate chains of higher sizes could exceed any certificate chain that our applications see today. That could mean many more packets which increases the loss probability in constrained conditions [29]. It also could lead to more round-trips due to TCP Congestion Control [42, 41] and, thus, connection establishment slowdowns.

Specifically (D)TLS includes multiple signatures in its handshake which could fluctuate based on usecase. All connections include a signature in the `CertificateVerify` message and public keys with signatures in each certificate in the chain. According to Shodan [40], ∼77% of TLS connections include certificate chains

with one or two ICA certificates, which usually do not exceed 4KB. X.509 leaf certificates used in the Web (HTTPS), on the other hand, usually include two or more additional Signed Certificate Timestamps (SCTs) [20] which incorporate one signature each. Recently, browsers have increased their minimum SCT requirement. For example, as of April 2021, Apple's Certificate Transparency policy [2] requires three SCTs if the certificate lifetime is longer than 180 days. Similarly, Chrome has been requiring at least two SCTs or more depending on certificate lifetime. If SCTs are not included in the certificate, they can optionally be included in the handshake in a TLS Extension. Furthermore, when Online Certificate Status Protocol (OCSP) [35] stapling is used in TLS, one more OCSP signature may be included in the handshake to verify certificates are not revoked. Thus, it is clear that (D)TLS can include $(x + 2) + s + o$ signatures and $x + 1$ public keys, where x is the number of ICAs in the certificate chain, s is the number of SCTs and $o = 1$ only if OCSP stapling is used.

To quantify the minimum authentication data size of PQ certificate authentication in (D)TLS, we calculated them for the leanest PQ signature candidates in NIST's Round 3. Lattice-based Dilithium and Falcon offer the smallest public key and signature sizes. Rainbow is the third PQ signature finalist and SPHINCS$^+$ is the most well-analyzed and trusted algorithm in terms of cryptographic primitives. We analyzed authentication data sizes for all the parameters of Dilithium and Falcon, and the leanest ones for Rainbow and SPHINCS$^+$. We assumed 500KB of X.509 attributes in each certificate. In terms of certificate formats, we assumed binary DER encoding [12] which is used to transfer certificates on the wire.

Table 1 includes our results. We can observe that only Falcon is consistently in the 4-8KB range for 1-4 ICAs which is in the range of the sizes we see today. Dilithium offers less flexibility and remains below ∼14.5KB, the most commonly used TCP `initcwnd` used today (10MSS), only for its Dilithium-2 parameter set for two or more ICAs in the chain. When SCTs and/or OCSP staples are present Dilithium starts from ∼15KB. Note that even below ∼14KB, the more data included in a handshake the higher the loss probability and the connection slowdown in lossy environments [29]. Based on [51], we would like to minimize this data to 9-10KB (in green), not just below ∼14.5KB. All other post-quantum signature algorithms are shown to exceed 15-20KB which admittedly is a higher price to pay in order to establish a connection that otherwise exchanges much less data.

QUIC [14, 48], on the other hand, is a protocol which was built with speed and performance in mind. It runs over UDP and uses TLS 1.3 for its secure tunnel negotiation. QUIC includes an amplification protection mechanism according to which initial QUIC client data has to be padded to at least 1200B and the server response should not exceed three times the initial client request size. If it does, the server has to wait for a response from the client (one round-trip) before sending more data. Amplification protection is used only for addresses which have not yet been validated.

	1 ICA	2 ICAs	3 ICAs	4 ICAs	PQ Signature
TLS (no SCTs, no OCSP staples)	8.77	11.94	15.12	18.29	Dilithium-2
	11.91	16.22	20.53	24.84	Dilithium-3
	3.76	5.31	6.86	8.40	Falcon-512
	6.64	9.32	12.00	14.68	Falcon-1024
	89.12	133.64	178.16	222.69	Rainbow-I(cz)
	20.44	26.73	33.02	39.31	SPHINCS$^+$-128s
Web TLS (SCTs, no OCSP staples)	12.40	15.57	18.75	21.92	Dilithium-2
	16.85	21.16	25.47	29.78	Dilithium-3
	4.76	6.31	7.85	9.40	Falcon-512
	8.56	11.24	13.92	16.60	Falcon-1024
	89.21	133.74	178.26	222.79	Rainbow-I(cz)
	32.22	38.51	44.80	51.10	SPHINCS$^+$-128s
Web TLS (Web (SCTs, OCSP staples)	14.82	17.99	21.17	24.34	Dilithium-2
	20.14	24.45	28.76	33.07	Dilithium-3
	5.43	6.97	8.52	10.07	Falcon-512
	9.84	12.52	15.20	17.88	Falcon-1024
	89.28	133.80	178.33	222.85	Rainbow-I(cz)
	40.08	46.37	52.66	58.95	SPHINCS$^+$-128s

Table 1: Approximate size of auth data (`CertificateVerify` + `Certificates`) in TLS handshake (KB)

In order to quantify the impact of PQ algorithms on QUIC's amplification protection, we analyzed the size of QUIC messages using PQ algorithm candidates in NIST's Round 3. For key exchange, we analyzed hybrid key exchange that uses X25519 [19] with a PQ KEM candidate NIST Round 3 finalist like Kyber, Saber or NTRU. We evaluated various parameters for these PQ KEMs. Regarding signatures, we chose the leanest lattice-based signature parameters (i.e., Dilithium-2, Falcon-512). We assumed no SCTs or OCSP stapling were used, 500B of X.509 attributes and the certificates were DER encoded.

Table 2 summarizes the QUIC message sizes. We observe that all of the PQ signature options will exceed (in red) 3× the initial request packet and will trigger QUIC's amplification protection which would lead to a slowdown due to the extra round-trip.

Initial Client Data (B)	Server Data (KB)			PQ Algorithms
	1 ICA	2 ICAs	3 ICAs	
1050	13.32	16.49	19.66	(X25519 + Kyber-512)+ Dilithium-2
1050	5.68	7.22	8.77	(X25519 + Kyber-512) + Falcon-512
949	5.61	7.16	8.70	(X25519 + ntruhps2048509) + Falcon-512
922	5.65	7.19	8.74	(X25519 + LightSaber) + Falcon-512
1434	6.00	7.54	9.09	(X25519 + Kyber-768) + Falcon-512
1181	5.84	7.39	8.94	(X25519 + ntruhps2048677) + Falcon-512
1242	6.00	7.54	9.09	(X25519 + Saber) + Falcon-512
1818	6.48	8.02	9.57	(X25519 + Kyber-1024) + Falcon-512
1480	6.14	7.69	9.23	(X25519 + ntruhps4096821) + Falcon-512
1562	6.38	7.93	9.48	(X25519 + FireSaber) + Falcon-512

Table 2: Approximate Initial Client and Server Data in QUIC using Hybrid Key Exchange and PQ signatures

From the data in Tables 1 and 2 it is clear that big certificate chains could pose challenges to secure tunnel establishment. EAP, constrained condition applications, and post-quantum (D)TLS and QUIC connections will be negatively affected. Our goal is to alleviate these challenges in order to speed up the secure tunnel handshakes.

3 Related Work

Various works have focused on the issues certificate chains introduce to certain usecases. IETF RFC drafts draft-ietf-emu-eaptlscert [39] and ietf-emu-eaptls13 [21] discuss the problem of big TLS certificate chains for EAP authentication. [15] presents the heavy authentication data issue with (D)TLS connections in Wi-SUN and IEEE 802.14.5 mesh networks. Compact TLS (cTLS) IETF draft draft-ietf-tls-ctls [31] is a compact version of TLS 1.3 designed for constrained conditions. cTLS proposes using pre-determined certificate dictionaries which peers can use to convey their certificate chains without actually sending the certificates. In the same context, Mozilla introduced an ICA Pre-load list from the multi-browser Common CA Database (CCADB) [25] in its Desktop Firefox browser in 2020.

On the size of post-quantum authentication data, early work by Bindel et al. emulated large hybrid PQ certificates and studied their impact on TLS libraries and browsers [3]. [16] showed that post-quantum Stateful Hash-Based Signatures in certificates will not break (D)TLS, QUIC and IKEv2. Sikeridis et al. [42] also studied the impact of PQ signatures on TLS 1.3 and proved that lattice-based PQ candidates offer the most efficient options whereas all other NIST Round 2 schemes could introduce round-trips due to the TCP `initcwnd`.

Additionally, [41] tested hybrid (i.e. classical ECDH and PQ KEMs) key exchange and signatures in TLS 1.3 and SSH. It showed that some lattice-based PQ algorithms schemes do not detrimentally slow down TLS handshakes. [51], on the other hand, showed that 9-10 KB certificate chains will lead to double digit TLS handshake slowdowns and some clients or middleboxes cannot handle chains larger than 10KB.

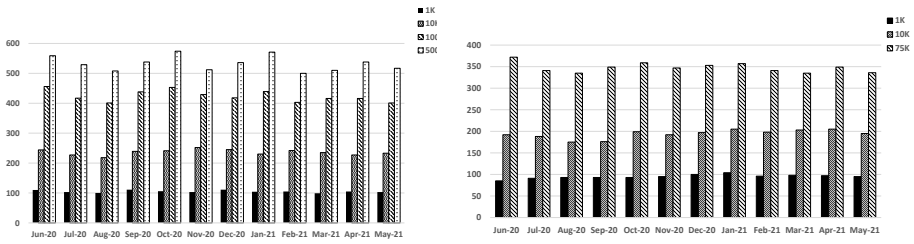
In [6], Crockett et al. presented the challenges of implementing NIST's PQ key exchange and authentication algorithms in TLS and SSH, with a focus on hybrid schemes. What's more, Paquin et al. showed in [29] that the more data included in a PQ handshake, the higher the loss probability ($1-(1-p)^n$, where n is the number of authentication packets and p is the individual packet loss probability) and the connection slowdown in unstable network environments. Finally, Schwabe et al. proposed KEMTLS [38] which uses PQ KEMs in the leaf certificate. One of the advantages it offers is that the certificate chain ends up being a few KB smaller than it would have been when using lattice-based PQ signatures.

4 ICA Statistics

From our analysis so far, it is clear that certificate chains could pose challenges to secure tunnel establishment for some of today's usecases and most future post-quantum ones. A straightforward, potential solution which we will investigate in Section 5 is caching ICA certificates and omitting them from the handshake. Before looking into solutions, we wanted to study how many ICA certificates exist today and how many someone would need to cache in order to speed up secure tunnel establishment.

Initially, we used CCADB's ICA list to get the number of non-revoked active ICA certificates used in the Web. We also used the Alexa and Cisco Umbrella Top1M datasets [1, 47] for the top visited sites. Alexa is a well-known ordered list of the most popular sites on the Internet. Since Alexa Top1M stopped being free, Cisco published their own dataset, Cisco Umbrella Top1M. The Umbrella dataset is different than Alexa's as it is built with a different methodology. In order to retrieve the certificates of the top visited sites in our two datasets, we used data from Censys.io [4]. Censys.io is a popular analytics engine which scans the Internet daily and inventories information about connections to public open servers.

We wanted to investigate the status of the certificate chains returned from servers over time. Thus, we analyzed the certificates returned from the Alexa and Umbrella sites at the beginning of each month for a period of 12 months (June 2020-May 2021). The results are shown in Table 3. We can see that the total non-revoked Web ICA certificates (based on CCADB [26]) were 1306 which roughly matches with ICA count for the Web in [50]. For the domains within the Top1M Alexa sites supporting HTTPS, the distinct ICAs were 500-560. For the domains that support HTTPS within the Top1M Umbrella sites, the ICAs were 335-375. Caching these ICA certificates is manageable for most usecases.



(a) Alexa (b) Umbrella

Fig. 2: # of ICAs for TopX servers (12 month)

Our results also show that a few of the top sites on the Internet do not include ICAs in their certificate chain; they either send only a server certificate or a server certificate and Root CA certificates. We also see that ∼99% of the sites include up to 3 ICA certificates and ∼96% (∼92% for Umbrella) include 1-2 ICA certificates. We also see a small amount of self-siged (SS) certificates. Self-signed certificates could be server or Root CA certificates. For the purpose of alleviating as much handshake data as possible for the (D)TLS handshake, Root CA certificates should not be sent[3]. We also observed a limited number of certificates which were not server certificates but also had the *BasicConstraints* X.509 extension set to *cA:False*. These were misconfigurations. Readers should note that Alexa's Top1M sites ended up being ∼500K and Umbrella's ∼75K servers. The reason is that we investigated servers that were listening on well-known TLS port 443.

Subsequently, we wanted to quantify the ICA certificates in our datasets for each month and how the count increased based on the top servers visited. Fig. 2 shows the ICA certificate count per month using data from Censys.io for the Top 1K, 10K, 100K, 500K servers for Alexa and 1K, 10K, 75K for Umbrella. Note that Umbrella's dataset included much less TCP port 443 active servers which is why the count is lower. We can see how the ICA count increases with the number of top servers. Intuitively that makes sense as the more servers one examines (using scanning or by directly querying databases such as Censys.io) the more ICAs are discovered and thus the less ICAs remain unseen. As we can see in the two figures, the ICA size growth slows down as the servers increase; the ICA counter increases faster between 1K and 10K servers. Most Internet peers tend to get certs from a limited set of CAs (e.g., Let's Encrypt CA) and not evenly use all ICAs available. Thus, the rate of increase of the ICA count is slower than the server count.

[3] Because certificate validation requires that root keys be distributed independently, the self-signed certificate that specifies the root certificate authority MAY be omitted from the chain, under the assumption that the remote end must already possess it in order to validate it in any case [7, 30]

Data Set	0 ICAs	1 ICA	2 ICAs	3 ICAs	¿3 ICAs	# distinct servers	SS certs	non-CA certs	Distinct ICAs
Alexa 06-2020	632	388071	82325	12418	579	484025	45738	3545	559
Alexa 07-2020	1339	350178	74475	11661	685	438338	39543	6104	529
Alexa 08-2020	629	318805	66524	11369	442	397769	38741	3252	508
Alexa 09-2020	582	323014	67606	11680	380	403262	37071	3057	538
Alexa 10-2020	816	480133	104064	16734	537	602284	58195	4258	574
Alexa 11-2020	498	339441	62940	12496	442	415817	41138	3177	512
Alexa 12-2020	1136	353411	73531	12878	580	441536	40064	6764	536
Alexa 01-2021	1350	414144	90538	15209	719	521960	46379	7177	571
Alexa 02-2021	1090	362439	75886	12994	693	453102	39501	6314	500
Alexa 03-2021	1006	364351	77777	12973	627	456734	41062	6024	510
Alexa 04-2021	809	469614	98482	17240	556	586701	50654	3714	538
Alexa 05-2021	607	440865	92548	13469	502	547991	45668	3166	517
Umbrella 06-2020	52	56981	13216	5171	47	75467	11795	465	372
Umbrella 07-2020	43	54480	11887	4860	54	71324	11040	425	341
Umbrella 08-2020	52	52741	11353	4877	44	69067	10780	424	335
Umbrella 09-2020	42	53744	11603	5042	44	70475	11070	411	349
Umbrella 10-2020	63	64214	14228	6113	71	84689	13261	570	359
Umbrella 11-2020	44	57246	11475	5554	53	74372	11316	458	347
Umbrella 12-2020	54	57844	11905	5482	63	75348	11117	478	353
Umbrella 01-2021	53	62994	13129	5829	47	82052	12395	538	357
Umbrella 02-2021	48	59890	11432	5560	49	76979	10624	504	341
Umbrella 03-2021	58	56084	10878	5263	34	72317	99994	31	335
Umbrella 04-2021	62	67689	14042	7044	65	88902	12748	569	349
Umbrella 05-2021	49	66686	13554	7020	61	87370	12122	542	336
Mozilla CCADB [26]									1306

Table 3: CA data from Alexa, Umbrella Top 1M (12 month) and Mozilla's CCADB (June 2021).

Date	**2020-06-02**	**2020-07-07**	**2020-08-04**	**2020-09-01**	**2020-10-06**	**2020-11-03**
Intersection	53909	49755	49414	49430	60569	51491
Kendall τ	0.258	0.226	0.221	0.247	0.241	0.256
Date	**2020-12-01**	**2021-01-01**	**2021-02-01**	**2021-03-01**	**2021-04-01**	**2021-05-01**
Intersection	53367	58347	54202	51822	62446	61465
Kendall τ	0.256	0.238	0.234	0.228	0.249	0.247

Table 4: Intersection of domains and ordinal dissimilarity (using the Kendall rank correlation coefficient) between the Alexa and Umbrella lists over time.

We also wanted to compare our datasets. Table 4 shows the intersection of domains between the Alexa and Umbrella lists. Specifically, the table tabulates the number of common domains supporting HTTPS/TLS found in Censys.io for the twelve days we examined. We observe that the two top lists are different; this is expected since they are compiled using different methodologies (see also [37, 46]). The table also illustrates the ordinal similarity between the two lists using the Kendall rank correlation coefficient (also employed in [37] for similar comparisons). Although the two top lists are dissimilar, the number of ICAs required to be cached does not differ much.

5 Speed Up Mechanisms

So far we have seen that data-heavy authentication could negatively affect secure tunnel establishment. Thus, limiting the authentication data in these connections could alleviate the issue. There are multiple ways to achieve that. The most straightforward one is to suppress the ICA certificates in the handshake after the peer has cached them and signalled that it does not need them. [18] proposes using a TLS 1.3 flag extension to request the peer to suppress its ICA ceritificates. A flag in the `ClientHello` and the `CertificateRequest` would suffice for the client or server to request ICA suppression. Suppressing ICAs would drop the authentication data in Table 1 to acceptable levels (9-11KB) even for Web TLS connections. It would also drop the server response sizes in Table 2 to ∼5.5 and 3.2KB for Dilithium-2 and Falcon-512 respectively. Extrapolating from [42, 41, 51], that would speed up the handshake by ∼60ms when TCP `initcwnd`=10MSS or ∼10% when `initcwnd`=30MSS.

Another option for TLS is to increase the TCP `initcwnd`. [41] showed that by generously increasing it, handshakes can be sped up by eliminating extra round-trips. Readers should note that this option does not prevent handshake slowdowns in environments with increased loss probability [29]. But even in non-lossy environments, [51] showed that 9-10 KB certificate chains could lead to double digit TLS handshake slowdowns even with 30MSS TCP `initcwnd`. Additionally, although some Content Deliver Network (CDN) providers optimize their infrastructures and increase the TCP `initcwnd`, generally increasing `initcwnd` without thorough experimentation could negatively affect constrained

usecases, slow links, cellular networks, bursty traffic patterns, and highly multiplexed links in developing regions [5, § Appendix A].

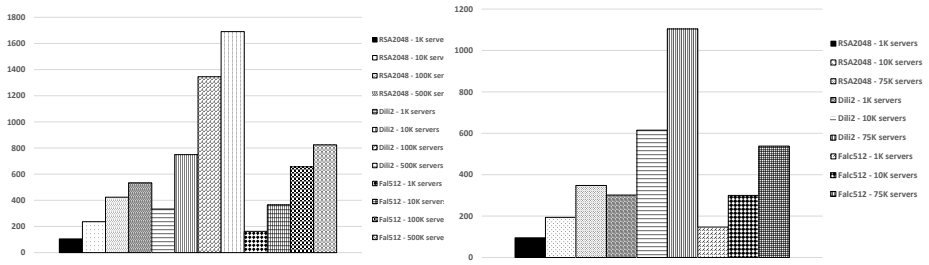
Alternatively, we could omit all certificates and use a fingerprint in the TLS handshake to indicate which peer certificate we have cached as proposed in [36]. Although standardized a few years already, this mechanism has not seen wide industry adoption. It also introduces security caveats and operational concerns like allowing TLS session correlation [36, § 7] and actively and frequently managing and updating a large certificate cache of certificates with relatively short lifetimes.

Section 2 discussed how QUIC amplification protection will introduce extra round-trips when using post-quantum signature algorithms. One option to prevent these round-trips is to include QUIC `PADDING` frames in the request in order for the response to fit within the 3× size of the request. We could pad with enough data to be safe in all cases, but then the client wastes resources unnecessarily without even knowing if a round-trip would be warranted based on the server's supported algorithms and certificates. Additionally, even if we prevented the round-trips, excessive authentication data will still be sent (in one round-trip) which still introduces increased loss probability in unstable or congested networks [29, 51].

[42, §VII-B] suggests using different signature mechanisms at the Root CA, the OCSP staple and the SCTs (for the Web) as a way to alleviate the data issue. The Root CA and the OCSP staple and SCT public keys are not sent in the handshake, thus using a signature algorithm that has a small signature would slim down the data sent. Example algorithms would be Stateful HBS signatures [22, 11] or multivariate candidates in NIST's PQ Project like Rainbow. Although this method would trim down the data, big signatures or public keys would still be included in the handshake, so the issue is not eliminated. And we should not underestimate that this option would require peers to support multiple signature algorithms. Introducing new algorithms has traditionally not been a smooth process for the industry.

cTLS [31] proposes using pre-established dictionaries to omit sending certificates in the handshakes. This method would work nicely for peers that can be provisioned with the right certificate dictionaries. Different usecases like the Web would pose challenges with establishing these dictionaries, keeping them up-to-date and making sure the peers have the same version.

From our analysis, it is obvious that the most straightforward option to convey to the (D)TLS peer to omit its ICA certificates is using a TLS 1.3 flag [27, 18]. This mechanism is backwards compatible; if the peer does not support the flag the connection still completes. It also would work well when we communicate with finite peers whose ICA certificates is trivial to cache. In usecases where there are multiple or infinite peers, we need an all-inclusive ICA list or an ICA caching mechanism which we discuss in Section 6.



(a) Alexa (b) Umbrella

Fig. 3: 12 month average ICA cache size (KB) for TopX servers

6 ICA Caching

After discussing the best options to alleviate the authentication-data issue, we wanted to quantify the size of the ICA cache that would suffice for clients visiting the Top1M servers in our datasets. We analyzed the ICA certificate cache size as the top server count in our datasets increases. Again, we assumed 500B of X.509 attributes in each binary DER encoded [12] certificate. We also assumed that the same algorithm is used for all signatures in the TLS handshake and that ICAs certificates do not contain SCTs.

Fig. 3 shows the ICA certificate sizes for RSA-2048, Dilithium-2 and Falcon-512 based on the average 12-month ICA certificate count from our Alexa and Umbrella datasets. We can see how the cache size for caching all ICAs increases as we include more servers from 1K to 500K for Alexa and from 1K to 75K for Umbrella. Intuitively that makes sense as more servers mean more CA vendors issuing their certificates. Fig. 3a shows that the ICA cache size for Alexa servers will not exceed 550KB on average for RSA-2048 ICAs, 1.7MB for Dilithium-2 and 850KB for Falcon-512 ICAs. Fig. 3b shows the equivalent sizes for Umbrella are 400KB, 1.1MB and 600KB. Although someone could use more efficient caching mechanisms than caching all ICAs, we believe this analysis shows that caching does not introduce detrimentally high resource requirements (<1-2MB) even for big post-quantum ICA certificates. For comparison, the Web cache size including all RSA and ECDSA ICAs chaining to Root CAs trusted in the Mozilla Root Program is ∼1MB [50] which is manageable for most applications.

Although we could probably cache all ICAs for some usecases, it would make more sense to limit our cache size especially for cases where an entity can communicate with infinite peers or where an up-to-date full ICA list is not available. A crude option could be to use a static partial list of common ICAs and only request suppression from peers we have seen before that use a chain with ICAs in that list. More efficient options are, of course, possible. Caching is a well-studied topic used in various Internet and memory usecases. Most mechanisms cache data and usually have a way to update the cache when there is a cache miss (missing entry in the cache).

For the purposes of ICA caching we can follow a similar approach. We initially have our ICA List which consists of the ICA certificates cached while

connecting with peers. These certificates can be omitted from subsequent connections. The ICA List entries are referenced by a secondary list (Peer List) which binds peers with the ICAs cached. The Peer List is an ordered list for faster lookups. It includes the ICAs in the peer's certificate chain, a counter of the times communicated with that peer, a timestamp for the last communication and a timeout value. These attributes will be used by the caching mechanism in order to update the cache. Fig 4 shows the ICA cache architecture.

The Peer List timeout is used to clean up the cache at regular maintenance intervals. It can be set according to a default timeout value, or it can be updated based on the frequency of a cache miss. Busy caches dealing with multiple peers are normally more frequently updated. Deciding on the best timeout value is a trade-off decision; the lower the timeout the more operational burden on the cache, the higher the timeout the more peer entry evictions will need to take place with a cache miss. Additionally, timeouts can be set based in certificate expiration. When adding a peer entry, we could set the timeout to the peer certificate expiration. When the peer certificate expires, it is expected that we would need a new connection to get its new certificate and ICAs. An ICA revocation would not affect security, as the ICA cache does not preclude normal certificate revocation checks when validating the peer identity.

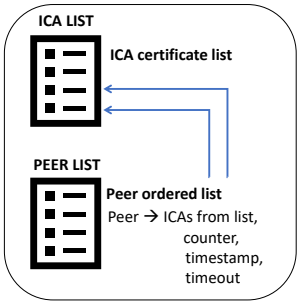


Fig. 4: ICA Cache

Algorithm 1 shows a simple cleanup mechanism based on one criterion. That could be the timeout in a peer entry in the Peer List which removes the peer from the list. Generally, if there are ICAs in the ICA List which are not referenced by any peer entries, these are deleted as proper cache hygiene.

```
for PeerEntry in Peer List do
  if (PeerEntry eviction criteria met) then
    Remove PeerEntry from Peer List
  end if
end for
for ICAEntry in ICA List do
  if (ICAEntry is not referenced by any peer in the Peer List) then
    Remove ICAEntry from ICA List
  end if
end for
```

Algorithm 1: ICA Cache Cleanup

Algorithm 2 shows the process of updating the ICA and Peer Lists when connecting to a peer. Before connecting to a new peer, we look up the Peer

List and if the peer already exists we update the counter and timestamp of the entry and connect by asking for ICA suppression. If the connection fails due to a certificate chain authentication error we remove that peer from the Peer List so that subsequent connections will not depend on the cached ICAs.

In case the peer does not exist in the Peer List, we connect without ICA suppression, we update the ICA cache with the peer's ICAs and add the peer in the Peer List. When updating the cache with new ICAs we can use Algorithm 1 if the cache is full. We could also remove peer entries and their corresponding ICAs from the two lists in order to make room by using different criteria like peer age (timestamps), counter, or randomly.

```
if (NewPeer in Peer List) then
  Update NewPeer entry counter, timestamp in Peer List
  Connect to NewPeer asking for ICA suppression
  if (Connection failed) then
    Remove NewPeer from Peer List
  end if
else
  Connect to NewPeer without ICA suppression and get NewPeer ICAs.
  while (not enough room in cache for NewPeer and its ICAs) do
    Make room for ICAs (Algorithm 1)
  end while
  Add NewPeer in Peer List
  Add NewPeer ICAs in ICA List
end if
```

Algorithm 2: New Connection ICA cache Update

Note that the caching mechanisms discussed do not require fault-tolerance, persistence or replication. They are built on straightforward concepts. In the event of a failure or application crash the cache can be rebuilt without having detrimental results other than potentially slower (due to heavy authentication-data) initial handshakes. Additionally, readers should note that the proposed ICA cache does not act as a CA store. These ICA certificates are cached, but not trusted by default in any way. Certificate chain validation does not change.

7 Considerations

Signalling to the peer that it ought to not send its ICA certificates is a straightforward option to trim data from the handshakes, but it comes with security considerations. When the client includes the TLS flag in its `ClientHello`, the flag is cleartext and passive observers could tell that the client is requesting ICA suppresion. As the feature would not be immediately ubiquitous in all clients, passive observers could use the signal to fingerprint clients and know which ones the traffic is coming from. If the TLS flag is included in the server

`CertificateRequest` which is an encrypted message in TLS 1.3 the concern does not apply.

Client fingerprinting is a security concern especially in a world where user privacy has become top of mind for users and vendors. We should note that the information leaked by the TLS flag is about the client having communicated with certain peers and about having added support for ICA suppression. Although these can be considered some loss of privacy, there is no additional leakage about the identity of the client or the server. To alleviate fingerprinting concerns, the client can encrypt its `ClientHello` using draft-ietf-tls-esni [33]. Although harder, a passive observer could still infer that the client asked for suppression if the server supports it by observing the amount of encrypted data sent from the server in the response. Wide-adoption of the suppression mechanism in TLS clients and servers would minimize the effect of any fingerprinting. Even then, an ICA system cache could create side-channels which could leak information (e.g., communicating destinations) even to off-path adversaries. Thus, the cache should be protected.

Based on the usecase, ICA suppression would be trivial when the amount of peers we are communicating with is finite or when the peers belong to a small set of PKI domains. For example, a device that only communicates with a controller and a few direct peers could trivially cache all ICAs. Usecases where the peers are infinite or the PKI domains numerous will need ICA caching mechanisms like the ones described in Section 6 when an up-to-date full ICA list is not available. Desktop Firefox preloading all Web ICAs [24] is another example where resourceful entities can store all ICAs that their peers could be using. Depending on how elaborate the caching mechanism is and how frequently cache misses take place, caching can be an operational concern. Designing proper caching is important for these cases especially since failures could have adverse effects on performance they are trying to improve.

Including one TLS flag for the client and one for the server to signal ICA suppression means that they can only ask for all or none of the ICA certificates from their peer. Sometimes ICA cache size is limited. In these cases we may want to control the ICA size. We may not want to cache ICAs that sign server certificates; we may want to start from second-level ICAs and above. We could achieve that by introducing one more TLS flags. Note that sometimes first or second-level ICAs are not clear-cut as an ICA may have signed leaf certificate and subordinate ICA certificates. To address that ambiguity in the caching algorithm, we would only depend on the peer certificate chain to decide the level of the ICA. We need detailed testing to decide if the complexity of another TLS flags is worth the cache size savings and the TLS handshake speedup.

The proposed caching mechanisms in Section 6 are based on straightforward practical approaches. Other options may exist. For example, a new ICA Chain List could include lists of pointers to ICA entries and a timeout or timestamp. The Peer List entries would then point to ICA Chain List entries. ICA List entries would be removed only when they are not referenced by any ICA Chain entry. Whole ICA Chain List entries would be removed when they are not referenced

by Peer List entries or when their timeout expires. In the event of full cache and a cache miss, ICA Chain entries, the corresponding peers and orphan ICAs would be removed to make room for new entries. Eviction mechanisms could include timeouts, age (timestamps) or counters in the ICA Chain entries.

As discussed in Section 6, the ICA cache does not operate like a CA Trust Store. Cached certificate authorities are not pre-trusted; they are only cached to avoid being sent on the wire. Someone may argue that the handshakes could be sped up by altering the chain validation to use the ICA cache entries as trusted certificates. There is a precedent with Trust Stores sometimes including ICAs and Root CAs. Ryan Sleevi explained in [44] the complication of overlapping certificate chains and how implementations suffer in correctly dealing with them. We consider pre-trusting any of the ICA cache certificates as more involved. We prefer solutions that align well with operations today and do not introduce new risks.

8 Conclusion and Future Work

In conclusion, in this work we saw that authentication data-heavy (D)TLS handshakes are slower. We quantified the issue and discussed usecases that are mostly affected by it. We evaluated potential alleviation mechanisms and argued that ICA suppression is the best option. We quantified the ICAs in use on the Internet today and we proved that ICA suppression can be made possible by caching ICAs. We also qualitatively evaluated ICA suppression signalling and caching mechanisms to make this possible. As future work, we are planning to investigate the expected ICA cache sizes for different usecases, applications and traffic profiles different that the Web. Finally, reviving [18] in IETF would be the next step to make authentication data-heavy (D)TLS connections lighter.

References

1. Amazon: Alexa top 1 Million (Aug 2021), `https://www.alexa.com/topsites/`
2. Apple: Apple's Certificate Transparency policy (Mar 2021), `https://support.apple.com/en-us/HT205280`
3. Bindel, N., Herath, U., McKague, M., Stebila, D.: Transitioning to a quantum-resistant public key infrastructure. In: Lange, T., Takagi, T. (eds.) Proc. 8th International Conference on Post-Quantum Cryptography (PQCrypto) 2017. LNCS, Springer (June 2017), to appear
4. Censys: censys.io data (Aug 2021), `https://censys.io/data`
5. Chu, J., Dukkipati, N., Cheng, Y., Mathis, M.: Increasing TCP's Initial Window. RFC 6928 (Apr 2013). https://doi.org/10.17487/RFC6928, `https://rfc-editor.org/rfc/rfc6928.txt`
6. Crockett, E., Paquin, C., Stebila, D.: Prototyping post-quantum and hybrid key exchange and authentication in TLS and SSH. In: NIST 2nd Post-Quantum Cryptography Standardization Conference 2019 (August 2019)
7. Dierks, T., Rescorla, E.: The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246 (Aug 2008). https://doi.org/10.17487/rfc5246, `https://rfc-editor.org/rfc/rfc5246.txt`

8. Fluhrer, S., Dang, Q.: Additional Parameter sets for LMS Hash-Based Signatures. Internet-Draft draft-fluhrer-lms-more-parm-sets-05, Internet Engineering Task Force (Jun 2021), https://datatracker.ietf.org/doc/html/draft-fluhrer-lms-more-parm-sets-05, work in Progress
9. Hoffman, P.E.: The Transition from Classical to Post-Quantum Cryptography. Internet-Draft draft-hoffman-c2pq-07, Internet Engineering Task Force (May 2020), https://datatracker.ietf.org/doc/html/draft-hoffman-c2pq-07, work in Progress
10. Housley, R.: Use of the HSS/LMS Hash-Based Signature Algorithm in the Cryptographic Message Syntax (CMS). RFC 8708 (Feb 2020). https://doi.org/10.17487/RFC8708, https://rfc-editor.org/rfc/rfc8708.txt
11. Huelsing, A., Butin, D., Gazdag, S.L., Rijneveld, J., Mohaisen, A.: XMSS: eXtended Merkle Signature Scheme. RFC 8391 (May 2018). https://doi.org/10.17487/RFC8391, https://rfc-editor.org/rfc/rfc8391.txt
12. International Telecommunications Union (ITU-T): ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)., https://www.itu.int/rec/T-REC-X.690-202102-I/en
13. International Telecommunications Union (ITU-T): 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
14. Iyengar, J., Thomson, M.: QUIC: A UDP-Based Multiplexed and Secure Transport. RFC 9000 (May 2021). https://doi.org/10.17487/RFC9000, https://rfc-editor.org/rfc/rfc9000.txt
15. Kampanakis, P., Chandra, R.: Mechanism to speed up secure communication handshakes in constrained conditions. Technical Disclosure Commons (Dec 2020), https://www.tdcommons.org/dpubs_series/3916/
16. Kampanakis, P., Panburana, P., Daw, E., Van Geest, D.: The Viability of Post-quantum X.509 Certificates. IACR Cryptology ePrint Archive **2018**, 63 (2018)
17. Kampanakis, P., Sikeridis, D.: Two pq signature use-cases: Non-issues, challenges and potential solutions. Cryptology ePrint Archive, Report 2019/1276 (2019), https://ia.cr/2019/1276
18. Kampanakis, P., Stebila, D., Friedl, M., Hansen, T., Sikeridis, D.: Post-quantum public key algorithms for the Secure Shell (SSH) protocol. Internet-Draft draft-kampanakis-curdle-pq-ssh-00, Internet Engineering Task Force (Oct 2020), https://datatracker.ietf.org/doc/html/draft-kampanakis-curdle-pq-ssh-00, work in Progress
19. Langley, A., Hamburg, M., Turner, S.: Elliptic Curves for Security. RFC 7748 (Jan 2016). https://doi.org/10.17487/RFC7748, https://rfc-editor.org/rfc/rfc7748.txt
20. Laurie, B., Langley, A., Kasper, E.: Certificate Transparency. RFC 6962 (Jun 2013). https://doi.org/10.17487/RFC6962, https://rfc-editor.org/rfc/rfc6962.txt
21. Mattsson, J.P., Sethi, M.: Using EAP-TLS with TLS 1.3 (EAP-TLS 1.3). Internet-Draft draft-ietf-emu-eap-tls13-18, Internet Engineering Task Force (Jul 2021), https://datatracker.ietf.org/doc/html/draft-ietf-emu-eap-tls13-18, work in Progress
22. McGrew, D., Curcio, M., Fluhrer, S.: Leighton-Micali Hash-Based Signatures. RFC 8554 (Apr 2019). https://doi.org/10.17487/RFC8554, https://rfc-editor.org/rfc/rfc8554.txt

23. Montenegro, G., Schumacher, C., Kushalnagar, N.: IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs): Overview, Assumptions, Problem Statement, and Goals. RFC 4919 (Aug 2007). https://doi.org/10.17487/rfc4919, https://rfc-editor.org/rfc/rfc4919.txt
24. Mozilla: Preloading Intermediate CA Certificates into Firefox (Nov 2020), https://blog.mozilla.org/security/2020/11/13/preloading-intermediate-ca-certificates-into-firefox/s
25. Mozilla: Common CA Database (CCADB) (Jul 2021), https://www.ccadb.org/resources
26. Mozilla: Common CA Database (CCADB) (Feb 2022), https://ccadb-public.secure.force.com/mozilla/MozillaIntermediateCertsCSVReport
27. Nir, Y.: A Flags Extension for TLS 1.3. Internet-Draft draft-ietf-tls-tlsflags-06, Internet Engineering Task Force (Jul 2021), https://datatracker.ietf.org/doc/html/draft-ietf-tls-tlsflags-06, work in Progress
28. Ounsworth, M., Pala, M.: Composite Signatures For Use In Internet PKI. Internet-Draft draft-ounsworth-pq-composite-sigs-05, Internet Engineering Task Force (Jul 2021), https://datatracker.ietf.org/doc/html/draft-ounsworth-pq-composite-sigs-05, work in Progress
29. Paquin, C., Stebila, D., Tamvada, G.: Benchmarking post-quantum cryptography in tls. In: Ding, J., Tillich, J.P. (eds.) Post-Quantum Cryptography. pp. 72–91. Springer International Publishing, Cham (2020)
30. Rescorla, E.: The Transport Layer Security (TLS) Protocol Version 1.3. RFC 8446 (Aug 2018). https://doi.org/10.17487/RFC8446, https://rfc-editor.org/rfc/rfc8446.txt
31. Rescorla, E., Barnes, R., Tschofenig, H.: Compact TLS 1.3. Internet-Draft draft-ietf-tls-ctls-03, Internet Engineering Task Force (Jul 2021), https://datatracker.ietf.org/doc/html/draft-ietf-tls-ctls-03, work in Progress
32. Rescorla, E., Modadugu, N.: Datagram Transport Layer Security Version 1.2. RFC 6347 (Jan 2012). https://doi.org/10.17487/rfc6347, https://rfc-editor.org/rfc/rfc6347.txt
33. Rescorla, E., Oku, K., Sullivan, N., Wood, C.A.: TLS Encrypted Client Hello. Internet-Draft draft-ietf-tls-esni-12, Internet Engineering Task Force (Jul 2021), https://datatracker.ietf.org/doc/html/draft-ietf-tls-esni-12, work in Progress
34. Rescorla, E., Tschofenig, H., Modadugu, N.: The Datagram Transport Layer Security (DTLS) Protocol Version 1.3. Internet-Draft draft-ietf-tls-dtls13-43, Internet Engineering Task Force (Apr 2021), https://datatracker.ietf.org/doc/html/draft-ietf-tls-dtls13-43, work in Progress
35. Santesson, S., Myers, M., Ankney, R., Malpani, A., Galperin, S., Adams, D.C.: X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP. RFC 6960 (Jun 2013). https://doi.org/10.17487/RFC6960, https://rfc-editor.org/rfc/rfc6960.txt
36. Santesson, S., Tschofenig, H.: Transport Layer Security (TLS) Cached Information Extension. RFC 7924 (Jul 2016). https://doi.org/10.17487/RFC7924, https://rfc-editor.org/rfc/rfc7924.txt
37. Scheitle, Q., Hohlfeld, O., Gamba, J., Jelten, J., Zimmermann, T., Strowes, S.D., Vallina-Rodriguez, N.: A long way to the top: Significance, structure, and stability of internet top lists. In: Proceedings of the Internet Measurement Conference 2018. p. 478–493. IMC '18, Association for Computing Machinery, New York, NY, USA (2018). https://doi.org/10.1145/3278532.3278574, https://doi.org/10.1145/3278532.3278574

38. Schwabe, P., Stebila, D., Wiggers, T.: Post-quantum tls without handshake signatures. In: Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security. p. 1461–1480. CCS '20, Association for Computing Machinery, New York, NY, USA (2020). https://doi.org/10.1145/3372297.3423350, https://doi.org/10.1145/3372297.3423350
39. Sethi, M., Mattsson, J.P., Turner, S.: Handling Large Certificates and Long Certificate Chains in TLS-based EAP Methods. Internet-Draft draft-ietf-emu-eaptlscert-08, Internet Engineering Task Force (Nov 2020), https://datatracker.ietf.org/doc/html/draft-ietf-emu-eaptlscert-08, work in Progress
40. SHODAN: HTTPS (443) Overview (2019), https://www.shodan.io/report/mNs9fa3I
41. Sikeridis, D., Kampanakis, P., Devetsikiotis, M.: Assessing the overhead of post-quantum cryptography in tls 1.3 and ssh. In: Proceedings of the 16th International Conference on Emerging Networking EXperiments and Technologies. p. 149–156. CoNEXT '20, Association for Computing Machinery, New York, NY, USA (2020). https://doi.org/10.1145/3386367.3431305, https://doi.org/10.1145/3386367.3431305
42. Sikeridis, D., Kampanakis, P., Devetsikiotis, M.: Post-quantum Authentication in TLS 1.3: A performance study. In: 27th Annual Network and Distributed System Security Symposium, NDSS 2020, San Diego, California, USA, February 23-26, 2020. The Internet Society (2020), https://www.ndss-symposium.org/ndss-paper/post-quantum-authentication-in-tls-1-3-a-performance-study/
43. Simon, D., Hurst, R., Aboba, D.B.D.: The EAP-TLS Authentication Protocol. RFC 5216 (Mar 2008). https://doi.org/10.17487/RFC5216, https://rfc-editor.org/rfc/rfc5216.txt
44. Sleevi, R.: Path Building vs Path Verifying: The Chain of Pain (Jun 2020), https://medium.com/@sleevi_/path-building-vs-path-verifying-the-chain-of-pain-9fbab861d7d6
45. Stebila, D., Fluhrer, S., Gueron, S.: Hybrid key exchange in TLS 1.3. Internet-Draft draft-ietf-tls-hybrid-design-03, Internet Engineering Task Force (Jul 2021), https://datatracker.ietf.org/doc/html/draft-ietf-tls-hybrid-design-03, work in Progress
46. Lists Study, T.: Scheitle, quirin and jelten, jonas (Jul 2021), https://toplists.github.io/index.html
47. Systems, C.: Cisco Umbrella 1 Million (Aug 2021), https://umbrella.cisco.com/blog/cisco-umbrella-1-million
48. Thomson, M., Turner, S.: Using TLS to Secure QUIC. RFC 9001 (May 2021). https://doi.org/10.17487/RFC9001, https://rfc-editor.org/rfc/rfc9001.txt
49. Tjhai, C., Tomlinson, M., Bartlett, G., Fluhrer, S., Geest, D.V., Garcia-Morchon, O., Smyslov, V.: Multiple Key Exchanges in IKEv2. Internet-Draft draft-ietf-ipsecme-ikev2-multiple-ke-03, Internet Engineering Task Force (Jul 2021), https://datatracker.ietf.org/doc/html/draft-ietf-ipsecme-ikev2-multiple-ke-03, work in Progress
50. Valsorda, F.: filippo.io/intermediates (Feb 2022), https://github.com/FiloSottile/intermediates
51. Westerbaan, B.: Sizing Up Post-Quantum Signatures (Nov 2021), https://blog.cloudflare.com/sizing-up-post-quantum-signatures/