Speeding Up Post-Quantum TLS handshakes by Suppressing Intermediate CA Certificates

PANOS KAMPANAKIS, AWS Cryptography, USA
MICHAEL KALLITSIS, Merit Network, Inc., USA

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 use cases are not significantly affected by big certificates or certificate chains, some constrained use cases such as low bandwidth environments or delay sensitive applications can see drastic performance degradation. 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 more amounts of post-quantum authentication data they include. One alleviation mechanism proposed in the literature is caching some of the authentication information. In this work, we 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 use cases 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.

Additional Key Words and Phrases: 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).

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). Following that, the Root CA is kept offline for security purposes. An ICA can further issue certificates for other ICAs that in turn sign leaf / entity certificates in the PKI. This process results to 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, 27, 29, 31], IKEv2/IPsec and QUIC [14, 44] 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 on the connection transcript which proves that it owns the identity private key corresponding to its identity certificate (Fig. 2). The server’s certificate is authenticated by verifying the certificate chain to a pre-trusted Root CA certificate. Including Root CA certificates in the certificate chain is only optional ([7, 27]) because they are pre-trusted and pre-installed on the peer. Given that certificate chains can be of arbitrary size, sometimes they could affect connection establishment 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 from censys.io [4] in order to quantify the number of ICAs used in (D)TLS connections today. We identify the ICAs, the distinct servers and the types of certificates sent in these 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 will not overload most software applications used today and would speed up the connection handshakes.

(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 ICA suppression signalling options for the peers to notify each other that they do not need the ICAs sent in the handshake. 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 protections mechanisms. Section 8 concludes this work and discusses future research topics.

2 THE HEAVY AUTHENTICATION DATA ISSUE

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 [21] usually operate in low-speed mediums. They often depend on EAP-TLS [39] for authentication. Establishing thousands of such simultaneous connections in a constrained medium can lead to contention and slow network authentication. Using big certificate chains in such usecases could further exacerbate the issue and lead to authentications that take minutes or hours [15, 19, 35].

Recent developments in quantum computing are expected to lead to further increases in the size of certificate chains which could slow down secure connection establishment even in typical Internet connections or applications. A quantum computer could solve (elliptic curve) discrete logarithm and integer factorization problems in polynomial time which could threaten all cryptography we use today. The cryptographic community has been researching quantum-secure public key algorithms for some time in order to address the quantum computer threat, and the US National Institute of Standards and Technology (NIST) has started a 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. A few Internet Engineering Task Force (IETF) RFC drafts are already introducing these algorithms in IETF standards [8–11, 20, 25, 41, 45]. 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 for the heavier schemes. The integration of such PQ signatures in X.509 certificates will naturally increase the size of these certificates significantly. 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 [26]. It also could lead to more round-trips due to TCP Congestion Control [37, 38] and, thus, connection establishment slowdowns.

TLS includes multiple signatures in its handshake which could fluctuate based on usecase. All TLS connections include a signature in the CertificateVerify and public keys with signatures in their certificate chains. According to Shodan [36], ~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 additional Signed Certificate Timestamps (SCTs) [18] 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) [32] is used in TLS, one more OCSP signature is included in the handshake. 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 of 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+. Readers should note that all other parameters and algorithm candidates in NIST’s Round 3 will exceed the chain sizes we estimated. 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. We also assumed that the same signature algorithm is used for all signatures in the TLS handshake. We analyzed the generic TLS usecase, Web connections with SCTs with and without OCSP staples for various chain lengths.

Table 1 includes our results. We can observe (in green) that only Falcon is consistently in the 4-8KB range for 1-4 ICAs which does not exceed the most commonly used ∼14.5KB (10MSS) TCP initcwnd used today and thus would not incur extra round-trips and slow down the handshake. Dilithium offers less flexibility and remains below ∼14KB only for its Dilithium-2 parameter set when more than one ICAs are 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 [26]. Thus we would like to minimize this data, not just keep it below \( \sim 14.5 \text{KB} \). 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, 44], 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. Given that QUIC uses UDP, it includes an amplification protection mechanism so that attackers could not spoof a small initial UDP packet and trigger an amplification attack where the server sends big chunks of data (including a certificate chain) to a victim client. Amplification protection defines that 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 (minimum 3.6KB total). If it does, the server has to wait for a response from the client (one round-trip) before sending more data. For most current key exchange and signature algorithms used in TLS 1.3 the \( 3 \times \) limit, as most certificate chains today do not exceed 3.5KB and would not trigger QUIC’s amplification protection. Readers should note that QUIC uses amplification protection only for addresses which have not yet been validated. After validating the address, the server does not enforce the amplification protection limit.

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 [17] 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 signatures parameters (i.e., Dilithium-2, Falcon-512). Using the leanest signatures offers the most balance possible between the QUIC initial client packet and the server response sizes. 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 can see that using any of the PQ KEM parameters with X25519 does not exceed 2KB. On the other hand, all of the PQ signature options (in red) will exceed \( 3 \times \) the initial request packet and will trigger QUIC’s amplification protection which would lead to a slowdown due to the extra round-trip.

<table>
<thead>
<tr>
<th>Initial Client Data (B)</th>
<th>Server Data (KB)</th>
<th>PQ Algorithms</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1 ICA</td>
<td>2 ICAs</td>
</tr>
<tr>
<td>1050</td>
<td>9.69</td>
<td>12.86</td>
</tr>
<tr>
<td>1050</td>
<td>4.68</td>
<td>6.23</td>
</tr>
<tr>
<td>949</td>
<td>4.61</td>
<td>6.16</td>
</tr>
<tr>
<td>922</td>
<td>4.65</td>
<td>6.19</td>
</tr>
<tr>
<td>1434</td>
<td>5.00</td>
<td>6.55</td>
</tr>
<tr>
<td>1181</td>
<td>4.84</td>
<td>6.39</td>
</tr>
<tr>
<td>1242</td>
<td>5.00</td>
<td>6.55</td>
</tr>
<tr>
<td>1818</td>
<td>5.48</td>
<td>7.03</td>
</tr>
<tr>
<td>1480</td>
<td>5.14</td>
<td>6.69</td>
</tr>
<tr>
<td>1562</td>
<td>5.38</td>
<td>6.93</td>
</tr>
</tbody>
</table>

Table 2. Approximate Initial Client and Server Data in QUIC using Hybrid Key Exchange and PQ signatures
From the data tabulated in Tables 1 and 2 it is clear that big certificate chains could pose challenges to secure tunnel establishment. In today’s world, they perform without major issues, but there are EAP and constrained condition applications which are negatively affected. In a PQ world, PQ signatures will introduce slowdowns in (D)TLS and QUIC handshakes almost for all of the NIST algorithm combinations getting standardized in NIST’s Round 3. 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-eap-tlscert [35] and ietf-emu-eap-tls13 [19] discuss the problem of big TLS cert chains for EAP authentication. They describe that authenticators sometimes drop an EAP session after only 40-50 round-trips which is a major deployment problem when big certificate chains are used in EAP-TLS. They also describe ways of alleviating the size of certificate chains. [15] presents the heavy authentication data issue with (D)TLS connections in Wi-SUN and IEEE 802.14.5 mesh networks which are significantly affected in constrained mediums where many devices are trying to join simultaneously and offers some ICA suppression techniques and caching mechanisms.

Compact TLS (cTLS) RFC draft draft-ietf-tls-ctls [28] is a compact version of TLS 1.3 which saves space by trimming obsolete material, tighter encoding, and a template-based specialization technique. It is 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 2020, Mozilla introduced an ICA Preload list from the multi-browser Common CA Database (CCADB) [23] in its Desktop Firefox browser. This list prevents inadvertent outages when connecting to misconfigured servers that do not return the right certificate chain in the TLS handshake. ICA Preloading also triggered IETF RFC draft draft-thomson-tls-sic [43] which proposes a new TLS 1.3 extension flag to signal ICA suppression to the peer.

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]. It proved that TLS implementations will still operate correctly in most cases. [16] showed that post-quantum Stateful Hash-Based Signatures in certificates will not break (D)TLS, QUIC and IKEv2. Sikeridis et al. [38] 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, [37] 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.

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. They showed that most PQ candidates can operate in TLS and SSH and identified some software challenges for some of the post-quantum schemes. What’s more, Paquin et al. showed in [26] 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 [34] 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.

Starting in 2020, Mozilla introduced an ICA Preload list from the multi-browser Common CA Database (CCADB) [23] in its Desktop Firefox browser. This list includes all ICAs used in Web certificate chains. Its goal is to prevent inadvertent outages when connecting to misconfigured servers that do not return the right certificate chain in the TLS handshake. Such servers sometimes return only the server certificate, or an incomplete, partial chain that is not rooted to a Root CA. Although the server certificate is a properly issued certificate that would otherwise be authenticated, such TLS stack misconfigurations lead browsers to fail authenticating the server. We used CCADB’s ICA list to get the number of non-revoked active ICA certificates used in the Web in July 2021.

We also used the Alexa and Cisco Umbrella Top1M datasets [1, 42] 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. Cisco argues it is more accurate. In order to retrieve the certificates of the top visited sites in our two datasets, we used censys.io’s dataset [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, not only for a snapshot in time. Thus, we analyzed the certificates returned from the Alexa, Umbrella sites at the beginning of each month for the period of 12 months (June 2020-May 2021). The results are shown in Table 3. We can see that the total non-revoked ICA certificates used on the Web (based on CCADB) is 4644. Desktop Firefox preloads all these certificates to prevent outages. Caching them all may not scale for smaller devices or post-quantum certificates which may be significantly bigger. Arguably, devices will not always connect to as many peers, thus the distinct ICA sets would be smaller in size. For example, Table 3 shows that for the Top1M Alexa sites, fluctuate between 500 and 560 ICAs over a period of 12 months. Caching $\sim500$ certificates is more manageable. For Umbrella, the equivalent was 335-375 ICAs.

Our results also show that a few of the top sites on the Internet do not include ICAs (0 ICAs) in their cert chain, they either send only a server certificate or server certificates and Root CA certificates. We also see that $\sim99\%$ of the sites include up to three ICA certificates and $\sim96\%$ ($\sim92\%$ for Umbrella) include one or two intermediate CA certificates. A site using more than three ICAs is rare. We also see a small amount of self-siged (SS) certificates. Self-siged certificates could be server or Root CA certificates. Root CA certificates in the certificate chain are only optional (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, 27]). For the purpose of alleviating as much handshake data as possible for the (D)TLS handshake, Root CA certificates should not be sent. We also saw a limited number of certificates which were not the server certificates but also had the BasicConstraints X.509 extension set to cA:False. These were neither peer identity certificates nor CA certificates. They were misconfigurations. Readers should note that Alexa’s Top1M sites ended up being $\sim500$K and Umbrellas $\sim75$K servers. The reason is that we investigated servers that were listening on well-known TLS port 443. Some of the sites in the lists were unreachable or were not listening for TLS on TCP 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. 3 shows the ICA certificate count per month from the Top 1K, 10K, 100K, 500K servers for Alexa and 1K, 10K, 75K for Umbrella. Note
<table>
<thead>
<tr>
<th>Data Set</th>
<th>0 ICAs</th>
<th>1 ICA</th>
<th>2 ICAs</th>
<th>3 ICAs</th>
<th>&gt;3 ICAs</th>
<th># distinct servers</th>
<th>SS certs</th>
<th>non-CA certs</th>
<th>Distinct ICAs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alexa 06-2020</td>
<td>632</td>
<td>388071</td>
<td>82325</td>
<td>12418</td>
<td>579</td>
<td>484025</td>
<td>45738</td>
<td>3545</td>
<td>559</td>
</tr>
<tr>
<td>Alexa 07-2020</td>
<td>1339</td>
<td>350178</td>
<td>74475</td>
<td>11661</td>
<td>685</td>
<td>438338</td>
<td>39543</td>
<td>6104</td>
<td>529</td>
</tr>
<tr>
<td>Alexa 08-2020</td>
<td>629</td>
<td>318805</td>
<td>66524</td>
<td>11369</td>
<td>442</td>
<td>397769</td>
<td>38741</td>
<td>3252</td>
<td>508</td>
</tr>
<tr>
<td>Alexa 09-2020</td>
<td>582</td>
<td>323014</td>
<td>67606</td>
<td>11680</td>
<td>380</td>
<td>403262</td>
<td>37071</td>
<td>3057</td>
<td>538</td>
</tr>
<tr>
<td>Alexa 10-2020</td>
<td>816</td>
<td>480133</td>
<td>104064</td>
<td>16734</td>
<td>537</td>
<td>602284</td>
<td>58195</td>
<td>4258</td>
<td>574</td>
</tr>
<tr>
<td>Alexa 11-2020</td>
<td>498</td>
<td>339441</td>
<td>62940</td>
<td>12496</td>
<td>442</td>
<td>415817</td>
<td>41138</td>
<td>3177</td>
<td>512</td>
</tr>
<tr>
<td>Alexa 12-2020</td>
<td>1136</td>
<td>353411</td>
<td>73531</td>
<td>12878</td>
<td>580</td>
<td>441536</td>
<td>40064</td>
<td>6764</td>
<td>536</td>
</tr>
<tr>
<td>Alexa 01-2021</td>
<td>1350</td>
<td>414144</td>
<td>90538</td>
<td>15209</td>
<td>719</td>
<td>521960</td>
<td>46379</td>
<td>7177</td>
<td>571</td>
</tr>
<tr>
<td>Alexa 02-2021</td>
<td>1090</td>
<td>362439</td>
<td>75886</td>
<td>12994</td>
<td>693</td>
<td>453102</td>
<td>39501</td>
<td>6314</td>
<td>500</td>
</tr>
<tr>
<td>Umbrella 06-2020</td>
<td>52</td>
<td>56981</td>
<td>13216</td>
<td>5171</td>
<td>47</td>
<td>75467</td>
<td>11795</td>
<td>465</td>
<td>372</td>
</tr>
<tr>
<td>Umbrella 07-2020</td>
<td>43</td>
<td>54480</td>
<td>11887</td>
<td>4860</td>
<td>54</td>
<td>71324</td>
<td>11040</td>
<td>425</td>
<td>341</td>
</tr>
<tr>
<td>Umbrella 08-2020</td>
<td>52</td>
<td>52741</td>
<td>11353</td>
<td>4877</td>
<td>44</td>
<td>69067</td>
<td>10780</td>
<td>424</td>
<td>335</td>
</tr>
<tr>
<td>Umbrella 09-2020</td>
<td>42</td>
<td>53744</td>
<td>11603</td>
<td>5042</td>
<td>44</td>
<td>70475</td>
<td>11070</td>
<td>411</td>
<td>349</td>
</tr>
<tr>
<td>Umbrella 10-2020</td>
<td>63</td>
<td>64214</td>
<td>14228</td>
<td>6113</td>
<td>71</td>
<td>84689</td>
<td>13261</td>
<td>570</td>
<td>359</td>
</tr>
<tr>
<td>Umbrella 11-2020</td>
<td>44</td>
<td>57246</td>
<td>11475</td>
<td>5554</td>
<td>53</td>
<td>74372</td>
<td>11316</td>
<td>458</td>
<td>347</td>
</tr>
<tr>
<td>Umbrella 12-2020</td>
<td>54</td>
<td>57844</td>
<td>11905</td>
<td>5482</td>
<td>63</td>
<td>75348</td>
<td>11117</td>
<td>478</td>
<td>353</td>
</tr>
<tr>
<td>Mozilla CCADB[23]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>4644</td>
</tr>
</tbody>
</table>

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

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 more servers mean more CAs issuing their certificates. The distinct ICA count for Alexa servers does not exceed 600 (Fig. 3a) and for Umbrella (Fig. 3b) 400. 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. That can be explained by the ICA distribution. We expect that the more servers from the dataset we examine, the more ICAs are included and thus the less ICAs we have not seen. 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 that the server count.
ICA Suppression

5 ICA SUPPRESSION 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. [43] proposes using a TLS 1.3 flag extension to request the peer to suppress its ICA certificates. [43]

struct {
    opaque flags<0..255>;
} FlagExtensions;

slightly deviates from [24] which defines the TLS 1.3 flags extension. [24] mandates that the flags used are unidirectional and the peer responds by setting the corresponding flag to acknowledge support, but [43] defines only one bidirectional flag for the client and server. Instead, a separate unidirectional flag in the ClientHello and the CertificateRequest would suffice for the client or server to request ICA suppression. The peer would acknowledge in its server or client Certificate message respectively and omit the ICAs. It is worth noting that [43] was an outcome of Mozilla’s ICA Preload work [23]. Other custom mechanisms of signalling to the peer to suppress its ICA certificate are also possible. Some have been documented in [15].

Another option for TLS is to increase the TCP initcwnd. [37] showed that by generously increasing the TCP initcwnd handshakes can be sped up by eliminating extra round-trips. Readers should note that this option does not prevent handshake slowdowns in lossy environments with increased loss probability \((1 - (1 - p)^n)\), where \(n\) is the number of authentication packets and \(p\) is the individual packet loss probability) as shown in [26]. 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 [33]. 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 [33, § 7] and actively and frequently managing and updating the certificate cache.

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. The client will usually not have knowledge of the server’s certificate chain size, thus it
could not be certain how much padding data it should include. There is always the option to pad
with enough data to be safe in all cases, but then the clients waste 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 [26].

[38, §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 [11, 20] or multivariate candidates in NIST’s PQ Project like Rainbow. Although this
method would slim down the data, big signatures or public keys would still be included in the
handshake, so the issue is not eliminated. Additionally, we can’t be certain if these algorithms will
be standardized by NIST. 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 [28] proposes using pre-established dictionaries to omit sending certificates in the hand-
shakes. 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 dic-
tionaries, keeping them up-to-date and making sure the peers have the same version. Also, a
third-party like CCADB would need to maintain and make these dictionaries available. Additionally,
dictionaries could serve as a fingerprinting mechanism for the client similar to HTTP User-Agent
string.

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 two separate TLS 1.3 flags [24, 43]. 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 ICA caching mechanism which
we discuss in Section 6.

6 ICA CACHING
After evaluating the best options to signal ICA suppression to the peer, we wanted to quantify the
size of the ICA cache that would suffice for clients visiting the Top1M servers in our datasets. To
do that 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 signature algorithm is used for all signatures in the TLS handshake and that
ICAs certificates do not contain SCTs.

Fig. 4 shows the ICA certificate sizes for RSA-2048 certificate, 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. Note that Umbrella’s dataset included much less TCP port
443 active servers which is why the sizes are smaller. Fig. 4a 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. 4b shows the equivalent sizes for Umbrella are 400KB, 1.1MB and
800KB. As we can see in the two figures, the ICA size growth slows down as the servers increase.
That can be explained by the ICA distribution. We expect the more servers from the dataset we
examine the more ICAs we include and thus the less ICAs we have not seen. Most servers on the Internet will tend to use a limited set of issuing CAs (e.g., Let’s Encrypt CA) and not evenly use all ICAs available. 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.

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 talk with infinite peers. 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 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 5 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. Algorithm 1 shows a simple cleanup mechanism based on a criteria. That could be the timeout in a peer entry in the Peer List which removes the peer from the list. If its ICAs are not referenced by any other active Peer entries, then the ICAs are removed from the ICA List as well. 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
    if (PeerEntry.icas not in any other Peer List icas) then
      Remove PeerEntry.icas from ICA List
    end if
    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 PeerEntry from Peer List
  end if
end for

Algorithm 1: ICA Cache Cleanup

Algorithms 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 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 remove peer entries and their corresponding ICAs from the two lists in order to make room by using different criteria like peer age (timestamps), or 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 ICAs in ICA List
  Add NewPeer in Peer 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, we are only caching some CAs used to build the authentications chain.

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 suppression. 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 [30]. draft-ietf-tls-esni uses public key encryption to encrypt the client's initial message which prevents a passive observer from seeing the TLS flag asking for ICA suppression. 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. Another option to prevent passive observers from fingerprinting the clients or servers by observing the amounts of data flowing through is randomizing the signal. Even if a client or server has cached ICAs about its peer, it could refrain from requesting ICA suppression for some of its requests. For example, it could probabilistically ask for suppression if the server supports it by observing the amount of encrypted data sent from the server. Another option to prevent passive observers from fingerprinting the clients or servers by observing the amounts of data flowing through is randomizing the signal. Depending on the probability distribution, that could prevent passive observers from successfully fingerprinting clients or servers. Choosing the right probability distribution while not significantly slowing down (D)TLS performance is a trade-off decision for the client or server.

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. Desktop Firefox preloading all Web ICAs [22] is another example where resourceful entities can store all ICAs that their peers could be using. Depending on how elaborate the algorithm is and how frequently cache misses take place, caching can be an operational concern. Designing proper caching is important for these cases.

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 would achieve that by introducing two more TLS flags, one for the client and one for the server. 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 two new TLS flags is worth the cache size saving 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 [40] 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 risky. We always prefer solutions that align well with operations today and do not introduce new risks or point of failure.

8 CONCLUSION AND FUTURE WORK

In conclusion, in this work we saw that authentication data-heavy (D)TLS handshakes are slower. We discussed usecases that are mostly affected by the issue like post-quantum (D)TLS and QUIC connections, EAP-TLS and constrained mesh networks. 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. We also want to experiment with some of the ICA caching mechanisms proposed in this work and their effect on (D)TLS handshake performance. Finally, reviving [43] in IETF would be the next step to alleviate authentication data-heavy (D)TLS connections.

REFERENCES


