Workgroup: Transport Layer Security
Internet-Draft: draft-ietf-tls-cert-abridge-01
Published: 16 March 2024
Intended Status: Experimental
Expires: 17 September 2024
Authors: D. Jackson
Mozilla

Abridged Compression for WebPKI Certificates

Abstract

This draft defines a new TLS Certificate Compression scheme which uses a shared dictionary of root and intermediate WebPKI certificates. The scheme smooths the transition to post-quantum certificates by eliminating the root and intermediate certificates from the TLS certificate chain without impacting trust negotiation. It also delivers better compression than alternative proposals whilst ensuring fair treatment for both CAs and website operators. It may also be useful in other applications which store certificate chains, e.g. Certificate Transparency logs.

About This Document

This note is to be removed before publishing as an RFC.

The latest revision of this draft can be found at https://tlswg.github.io/draft-ietf-tls-cert-abridge/draft-ietf-tls-cert-abridge.html. Status information for this document may be found at https://datatracker.ietf.org/doc/draft-ietf-tls-cert-abridge/.

Discussion of this document takes place on the Transport Layer Security Working Group mailing list (mailto:tls@ietf.org), which is archived at https://mailarchive.ietf.org/arch/browse/tls/. Subscribe at https://www.ietf.org/mailman/listinfo/tls/.

Source for this draft and an issue tracker can be found at https://github.com/tlswg/draft-ietf-tls-cert-abridge.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 17 September 2024.

Copyright Notice

Copyright (c) 2024 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (https://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Revised BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.

Table of Contents

1. Introduction

1.1. Motivation

When a server responds to a TLS Client Hello, the size of its initial flight of packets is limited by the underlying transport protocol. If the size limit is exceeded, the server must wait for the client to acknowledge receipt before concluding the flight, incurring the additional latency of a round trip before the handshake can complete. For TLS handshakes over TCP, the size limit is typically around 14,500 bytes. For TLS handshakes in QUIC, the limit is much lower at a maximum of 4500 bytes ([RFC9000], Section 8.1).

The existing compression schemes used in [TLSCertCompress] have been shown to deliver a substantial improvement in QUIC handshake latency [FastlyStudy], [QUICStudy] by reducing the size of server's certificate chain and so fitting the server's initial messages within a single flight. However, in a post-quantum setting, the signatures and public keys used in a TLS certificate chain will be typically 10 to 40 times their current size and cannot be compressed with existing TLS Certificate Compression schemes because most of the size of the certificate is in high entropy fields such as cryptographic keys and signatures.

Consequently studies [SCAStudy] [PQStudy] have shown that post-quantum certificate transmission becomes the dominant source of latency in PQ TLS with certificate chains alone expected to exceed even the TCP initial flight limit. This motivates alternative designs for reducing the wire size of post-quantum certificate chains.

1.2. Overview

This draft introduces a new TLS certificate compression scheme which is intended specifically for WebPKI applications and is negotiated using the existing certificate compression extension described in [TLSCertCompress]. It uses a predistributed dictionary consisting of all intermediate and root certificates contained in the root stores of major browsers which is sourced from the Common CA Database [CCADB]. As of May 2023, this dictionary would be 3 MB in size and consist of roughly 2000 intermediate certificates and 200 root certificates. The disk footprint can be reduced to near zero as many clients (such as Mozilla Firefox & Google Chrome) are already provisioned with their trusted intermediate and root certificates for compatibility and performance reasons.

Using a shared dictionary allows for this compression scheme to deliver dramatically more effective compression than previous schemes, reducing the size of certificate chains in use today by ~75%, significantly improving on the ~25% reduction achieved by existing schemes. A preliminary evaluation (Section 4) of this scheme suggests that 50% of certificate chains in use today would be compressed to under 1000 bytes and 95% to under 1500 bytes. Similarly to [SCA], this scheme effectively removes the CA certificates from the certificate chain on the wire but this draft achieves a much better compression ratio, since [SCA] removes the redundant information in chain that existing TLS Certificate Compression schemes exploit and is more fragile in the presence of out of sync clients or servers.

Note that as this is only a compression scheme, it does not impact any trust decisions in the TLS handshake. A client can offer this compression scheme whilst only trusting a subset of the certificates in the CCADB certificate listing, similarly a server can offer this compression scheme whilst using a certificate chain which does not chain back to a WebPKI root. Furthermore, new root certificates are typically included in the CCADB at the start of their application to a root store, a process which typically takes more than a year. Consequently, applicant root certificates can be added to new versions of this scheme ahead of any trust decisions, allowing new CAs to compete on equal terms with existing CAs as soon as they are approved for inclusion in a root program. As a result this scheme is equitable in so far as it provides equal benefits for all CAs in the WebPKI, doesn't privilege any particular end-entity certificate or website and allows WebPKI clients to make individual trust decisions without fear of breakage.

1.3. Relationship to other drafts

This draft defines a certificate compression mechanism suitable for use with TLS Certificate Compression [TLSCertCompress].

The intent of this draft is to provide an alternative to CA Certificate Suppression [SCA] as it provides a better compression ratio, can operate in a wider range of scenarios (including out of sync clients or servers) and doesn't require any additional error handling or retry mechanisms.

CBOR Encoded X.509 (C509) [I-D.ietf-cose-cbor-encoded-cert-05] defines a concise alternative format for X.509 certificates. If this format were to become widely used on the WebPKI, defining an alternative version of this draft specifically for C509 certificates would be beneficial.

Compact TLS, (cTLS) [I-D.ietf-tls-ctls-08] defines a version of TLS1.3 which allows a pre-configured client and server to establish a session with minimal overhead on the wire. In particular, it supports the use of a predefined list of certificates known to both parties which can be compressed. However, cTLS is still at an early stage and may be challenging to deploy in a WebPKI context due to the need for clients and servers to have prior knowledge of handshake profile in use.

TLS Cached Information Extension [RFC7924] introduced a new extension allowing clients to signal they had cached certificate information from a previous connection and for servers to signal that the clients should use that cache instead of transmitting a redundant set of certificates. However this RFC has seen little adoption in the wild due to concerns over client privacy.

Handling long certificate chains in TLS-Based EAP Methods [RFC9191] discusses the challenges of long certificate chains outside the WebPKI ecosystem. Although the scheme proposed in this draft is targeted at WebPKI use, defining alternative shared dictionaries for other major ecosystems may be of interest.

1.4. Status

This draft is still at an early stage. Open questions are marked with the tag **DISCUSS**.

2. Conventions and Definitions

The key words "**MUST**", "**MUST NOT**", "**REQUIRED**", "**SHALL**", "**SHALL NOT**", "**SHOULD**", "**SHOULD NOT**", "**RECOMMENDED**", "**NOT RECOMMENDED**", "**MAY**", and "**OPTIONAL**" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

This draft refers to dates in Internet Date/Time Format as specified in Section 5.6 of [DATES] without the optional T separator.

3. Abridged Compression Scheme

This section describes a compression scheme suitable for compressing certificate chains used in TLS. The scheme is defined in two parts. An initial pass compressing known intermediate and root certificates and then a subsequent pass compressing the end-entity certificate.

The compression scheme in this draft has two parameters listed below which influence the construction of the static dictionary. Future versions of this draft would use different parameters and so construct different dictionaries which would be registered under

different TLS Certificate Compression code points. This is discussed further in Section 5.

*CCADB_SNAPSHOT_TIME - 2023-01-01 00:00:00Z

*CT_CERT_WINDOW - 2022-12-01 00:00:00Z to 2023-01-01 00:00:00Z

3.1. Pass 1: Intermediate and Root Compression

This pass relies on a shared listing of intermediate and root certificates known to both client and server. As many clients (e.g. Mozilla Firefox and Google Chrome) already ship with a list of trusted intermediate and root certificates, this pass allows their existing lists to be reused, rather than requiring them to have to be duplicated and stored in a separate format. The first subsection details how the certificates are enumerated in an ordered list. This ordered list is distributed to client and servers which use it to compress and decompress certificate chains as detailed in the subsequent subsection.

3.1.1. Enumeration of Known Intermediate and Root Certificates

The Common CA Database [CCADB] is operated by Mozilla on behalf of a number of Root Program operators including Mozilla, Microsoft, Google, Apple and Cisco. The CCADB contains a listing of all the root certificates trusted by these root programs, as well as their associated intermediate certificates and not yet trusted certificates from new applicants to one or more root programs.

At the time of writing, the CCADB contains around 200 root program certificates and 2000 intermediate certificates which are trusted for TLS Server Authentication, occupying 3 MB of disk space. The listing used in this draft will be the relevant certificates included in the CCADB at CCADB_SNAPSHOT_TIME.

As entries on this list typically have a lifespan of 10+ years and new certificates are added to the CCADB a year or or more before being marked as trusted, future drafts which include newer certificates will only need to be issued infrequently. This is discussed further in Section 5.

The algorithm for enumerating the list of compressible intermediate and root certificates is given below:

1. Query the CCADB for all known root and intermediate certificates [CCADBAllCerts] as of CCADB_SNAPSHOT_TIME

2. Remove all certificates which have an extendedKeyUsage extension but do not have the TLS Server Authentication bit set or the anyExtendedKeyUsage bit set.

3. Remove all certificates whose notAfter date is on or before CCADB_SNAPSHOT_TIME.

4. Remove all root certificates which are not marked as trusted or in the process of applying to be trusted by at least one of the following browser root programs: Mozilla, Google, Microsoft, Apple.

5. Remove all intermediate certificates which do not chain back to root certificates still in the listing.

6. Remove any certificates which are duplicates (have the same SHA256 certificate fingerprint)

7. Order the list of certificates by the timestamp for when each was added to the CCADB, breaking any ties with the lexicographic ordering of the SHA256 certificate fingerprint.

8. Associate each element of the list with the concatenation of the constant 0xff and its index in the list represented as a uint16.

[[**DISCUSS:** The four programs were selected because they represent certificate consumers in the CCADB. Are there any other root programs which ought to be included? The only drawback is a larger disk requirement, since this compression scheme does not impact trust decisions.]]

[[**TODO:** Ask CCADB to provide an authoritative copy of this listing. A subset of these lists is available in benchmarks/data in this draft's repository.]]

3.1.2. Compression of CA Certificates in Certificate Chain

Compression Algorithm:

*Input: The byte representation of a Certificate message as defined in [TLS13] whose contents are X509 certificates.

*Output: opaque bytes suitable for transmission in a CompressedCertificate message defined in [TLSCertCompress].

1. Parse the message and extract a list of CertificateEntrys, iterate over the list.

2. Check if cert_data is bitwise identical to any of the known intermediate or root certificates from the listing in the previous section.

    1. If so, replace the opaque cert_data member of CertificateEntry with its adjusted three byte identifier

and copy the CertificateEntry structure with corrected lengths to the output.

2. Otherwise, copy the CertificateEntry entry to the output without modification.

3. Correct the length field for the Certificate message.

The resulting output should be a well-formatted Certificate message payload with the recognized intermediate and root certificates replaced with three byte identifiers and resulting lengths corrected. Note that the extensions field in each CertificateEntry remains unchanged, as does the certificate_request_context and any unrecognized certificates.

The decompression algorithm requires the above steps but in reverse, swapping any recognized three-byte identifier in a cert_data field with the DER representation of the associated certificate and updating the lengths. Unrecognized three-byte identifiers are ignored. Note that this does not have security implications, as the peer could send a Certificate message with an arbitrary payload directly. If the compressed certificate chain cannot be parsed (e.g. due to incorrect length fields) the decompression algorithm **MUST** report the failure and as required by [TLSCertCompress], the connection **MUST** be terminated with the "bad_certificate" alert.

TLS implementations intending to only use this scheme as a compressor (e.g. servers) **SHOULD** minimize the storage requirements of pass 1 by using a lookup table which maps the cryptographic hash of each certificate in the pass 1 listing to its assigned three byte identifier. This avoids the need for the compressor to retain a full copy of the pass 1 list. The hashing algorithm used in this lookup table is internal to the implementation and not exposed, but **MUST** be cryptographically secure. Note that implementations using this scheme as a decompressor (e.g. clients) typically already ship with a listing of trusted root and intermediate certificates which can be reused by the decompressor without any additional storage overhead.

3.2. Pass 2: End-Entity Compression

This section describes a pass based on Zstandard [ZSTD] with application-specified dictionaries. The dictionary is constructed with reference to the list of intermediate and root certificates discussed earlier in Section 3.1.1, as well as several external sources of information.

[[**DISCUSS:** This draft is unopinionated about the underlying compression scheme is used as long as it supports application specified dictionaries. Is there an argument for using a different scheme?]]

3.2.1. Construction of Shared Dictionary

[[**DISCUSS / TODO:** This section remains a work in progress and needs refinement. The goal is to produce a dictionary of minimal size which provides maximum compression whilst treating every CA equitably. Currently this dictionary occupies ~65KB of space, is equitable and has performance within a ~100 bytes of the best known alternative. This is discussed further in Section 4.]]

The dictionary is built by systematic combination of the common strings used in certificates by each issuer in the known list described in Section 3.1.1. This dictionary is constructed in three stages, with the output of each stage being concatenated with the next. Implementations of this scheme need only consume the finished dictionary and do not need to construct it themselves.

* Firstly, for each intermediate certificate enumerated in the listing in Section 3.1.1., extract the issuer field (Section 4.1.2.4 of [RFC5280]) and derive the matching authority key identifier (Section 4.2.1.1 of [RFC5280]) for the certificate. Order them according to the listing in Section 3.1.1, then copy them to the output.

* Secondly, take the listing of certificate transparency logs trusted by the root programs selected in Section 3.1.1, which are located at[AppleCTLogs] [GoogleCTLogs] as of CCADB_SNAPSHOT_TIME and extract the list of log identifiers. Remove duplicates and order them lexicographically, then copy them to the output.

* Finally, enumerate all certificates contained within certificate transparency logs above and witnessed during CT_CERT_WINDOW. For each issuer in the listing in Section 3.1.1, select the first end-entity certificate when ordered by the log id (lexicographically) and latest witnessed date.

  Extract the contents of the following extensions from the end-entity certificate selected for each issuer:

  - Authority Information Access (Section 4.2.2.1 of [RFC5280])

  - Certificate Policies (Section 4.2.1.4 of [RFC5280])

  - CRL Distribution Points (Section 4.2.1.13 of [RFC5280])

  - Freshest CRL (Section 4.2.1.15 of [RFC5280])

  Concatenate the byte representation of each extension (including extension id and length) and copy it to the output. If no end-entity certificate can be found for an issuer with this process, omit the entry for that issuer.

[[**DISCUSS:** This last step is picking a single certificate issued by each issuer as a canonical reference to use for compression. The ordering chosen allows the dictionary builder to stop traversing CT as soon as they've found an entry for each issuer. It would be much more efficient to just ask CAs to submit this information to the CCADB directly.]]

3.2.1.1. Compression of End-Entity Certificates in Certificate Chain

The resulting bytes from Pass 1 are passed to ZStandard [ZSTD] with the dictionary specified in the previous section. It is **RECOMMENDED** that the compressor (i.e. the server) use the following parameters:

- chain_log=30
- search_log=30
- hash_log=30
- target_length=6000
- threads=1
- compression_level=22
- force_max_window=1

These parameters are recommended in order to achieve the best compression ratio however implementations **MAY** use their preferred parameters as long as the resulting output can be decompressed by a [ZSTD]-compliant implementation. With TLS Certificate Compression, the server needs to only perform a single compression at startup and cache the result, so optimizing for maximal compression is recommended. The client's decompression speed is insensitive to these parameters.

4. Preliminary Evaluation

[[**NOTE:** This section to be removed prior to publication.]]

The storage footprint refers to the on-disk size required for the end-entity dictionary. The other columns report the 5th, 50th and 95th percentile of the resulting certificate chains. The evaluation set was a ~75,000 certificate chains from the Tranco list using the python scripts in the draft's Github repository.

Scheme	Storage Footprint	p5	p50	p95
Original	0	2308	4032	5609
TLS Cert Compression	0	1619	3243	3821

Scheme	Storage Footprint	p5	p50	p95
Intermediate Suppression and TLS Cert Compression	0	1020	1445	3303
This Draft	65336	661	1060	1437
This Draft with opaque trained dictionary	3000	562	931	1454
Hypothetical Optimal Compression	0	377	742	1075

Table 1

*'Original' refers to the sampled certificate chains without any compression.

*'TLS Cert Compression' used ZStandard with the parameters configured for maximum compression as defined in [TLSCertCompress].

*'Intermediate Suppression and TLS Cert Compression' was modelled as the elimination of all certificates in the intermediate and root certificates with the Basic Constraints CA value set to true. If a cert chain included an unrecognized certificate with CA status, then no CA certificates were removed from that chain. The cert chain was then passed to 'TLS Cert Compression' as a second pass.

*'This Draft with opaque trained dictionary' refers to pass 1 and pass 2 as defined by this draft, but instead using a 3000 byte dictionary for pass 2 which was produced by the Zstandard dictionary training algorithm. This illustrates a ceiling on what ought to be possible by improving the construction of the pass 2 dictionary in this document. However, using this trained dictionary directly will not treat all CA's equitably, as the dictionary will be biased towards compressing the most popular CAs more effectively.

*'Hypothetical Optimal Compression' is the resulting size of the cert chain after reducing it to only the public key in the end-entity certificate, the CA signature over the EE cert, the embedded SCT signatures and a compressed list of domains in the SAN extension. This represents the best possible compression as it entirely removes any CA certs, identifiers, field tags and lengths and non-critical extensions such as OCSP, CRL and policy extensions.

5. Deployment Considerations

5.1. Dictionary Versioning

The scheme defined in this draft is deployed with the static dictionaries constructed from the parameters listed in Section 3 fixed to a particular TLS Certificate Compression code point.

As new CA certificates are added to the CCADB and deployed on the web, new versions of this draft would need to be issued with their own code point and dictionary parameters. However, the process of adding new root certificates to a root store is already a two to three year process and this scheme includes untrusted root certificates still undergoing the application process in its dictionary. As a result, it would be reasonable to expect a new version of this scheme with updated dictionaries to be issued at most once a year and more likely once every two or three years.

A more detailed analysis and discussion of CA certificate lifetimes and root store operations is included in Appendix B, as well as an alternative design which would allow for dictionary negotiation rather than fixing one dictionary per code point.

[[**DISCUSS:** Are there concerns over this approach? Would using at most one code point per year be acceptable? Currently 3 of the 16384 'Specification Required' IANA managed code points are in use.]]

5.2. Version Migration

As new versions of this scheme are specified, clients and servers would benefit from migrating to the latest version. Whilst servers using CA certificates outside the scheme's listing can still offer this compression scheme and partially benefit from it, migrating to the latest version ensures that new CAs can compete on a level playing field with existing CAs. It is possible for a client or server to offer multiple versions of this scheme without having to pay twice the storage cost, since the majority of the stored data is in the pass 1 certificate listing and the majority of certificates will be in both versions and so need only be stored once.

Clients and servers **SHOULD** offer the latest version of this scheme and **MAY** offer one or more historical versions. Although clients and servers which fall out of date will no longer benefit from the scheme, they will not suffer any other penalties or incompatibilities. Future schemes will likely establish recommended lifetimes for sunsetting a previous version and adopting a new one.

As the majority of clients deploying this scheme are likely to be web browsers which typically use monthly release cycles (even long term support versions like Firefox ESR offer point releases on a monthly

basis), this is unlikely to be a restriction in practice. The picture is more complex for servers as operators are often to reluctant to update TLS libraries, but as a new version only requires changes to static data without any new code and would happen infrequently, this is unlikely to be burdensome in practice.

5.3. Disk Space Requirements

Clients and servers implementing this scheme need to store a listing of root and intermediate certificates for pass 1, which currently occupies around ~3 MB and a smaller dictionary on the order of ~100 KB for pass 2. Clients and servers offering multiple versions of this scheme do not need to duplicate the pass 1 listing, as multiple versions can refer to same string.

As popular web browsers already ship a complete list of trusted intermediate and root certificates, their additional storage requirements are minimal. Servers offering this scheme are intended to be 'full-fat' web servers and so typically have plentiful storage already. This draft is not intended for use in storage-constrained IoT devices, but alternative versions with stripped down listings may be suitable.

[[**DISCUSS:** The current draft priorities an equitable treatment for every recognized and applicant CA over minimizing storage requirements. The required disk space could be significantly reduced by only including CAs which meet a particular popularity threshold via CT log sampling.]]

5.4. Implementation Complexity

Although much of this draft is dedicated to the construction of the certificate list and dictionary used in the scheme, implementations are indifferent to these details. Pass 1 can be implemented as a simple string substitution and pass 2 with a single call to permissively licensed and widely distributed Zstandard implementations such as [ZstdImpl]. Future versions of this draft which vary the dictionary construction then only require changes to the static data shipped with these implementations and the use of a new code point.

There are several options for handling the distribution of the associated static data. One option is to distribute it directly with the TLS library and update it as part of that library's regular release cycle. Whilst this is easy for statically linked libraries written in languages which offer first-class package management and compile time feature selection (e.g. Go, Rust), it is trickier for dynamically linked libraries who are unlikely to want to incur the increased distribution size. In these ecosystems it may make sense to

distribute the dictionaries are part of an independent package managed by the OS which can be discovered by the library at run-time. Another promising alternative would be to have existing automated certificate tooling provision the library with both the full certificate chain and multiple precompressed chains during the certificate issuance / renewal process.

6. Security Considerations

This draft does not introduce new security considerations for TLS, except for the considerations already identified in [TLSCertCompress], in particular:

* The decompressed Certificate message **MUST** be processed as if it were encoded without being compressed in order to ensure parsing and verification have the same security properties as they would in TLS normally.

* Since Certificate chains are presented on a per-server-name or per-user basis, a malicious application cannot introduce individual fragments into the Certificate message in order to leak information by modifying the plaintext.

Further, implementors **SHOULD** use a memory-safe language to implement this compression schemes.

Note that as this draft specifies a compression scheme, it does not impact the negotiation of trust between clients and servers and is robust in the face of changes to CCADB or trust in a particular WebPKI CA. The client's trusted list of CAs does not need to be a subset or superset of the CCADB list and revocation of trust in a CA does not impact the operation of this compression scheme. Similarly, servers who use roots or intermediates outside the CCADB can still offer and benefit from this scheme.

7. Privacy Considerations

Some servers may attempt to identify clients based on their TLS configuration, known as TLS fingerprinting [FingerprintingPost]. If there is significant diversity in the number of TLS Certificate Compression schemes supported by clients, this might enable more powerful fingerprinting attacks. However, this compression scheme can be used by a wide range of clients, even if they make different or contradictory trust decisions and so the resulting diversity is expected to be low.

In TLS1.3, the extension carrying the client's supported TLS Certificate Compression schemes is typically transmitted unencrypted and so can also be exploited by passive network observers in addition to the server with whom the client is communicating. Deploying

Encrypted Client Hello [ECH] enables the encryption of the Client Hello and the TLS Certificate Compression extension within it which can mitigate this leakage.

8. IANA Considerations

[[**TODO:** Adopt an identifier for experimental purposes.]]

9. References

9.1. Normative References

[AppleCTLogs] Apple, "Certificate Transparency Logs trusted by Apple", 5 June 2023, <https://valid.apple.com/ct/log_list/current_log_list.json>.

[CCADBAllCerts] Mozilla, Microsoft, Google, Apple, and Cisco, "CCADB Certificates Listing", 5 June 2023, <https://ccadb.my.salesforce-sites.com/ccadb/AllCertificateRecordsCSVFormat>.

[DATES] Klyne, G. and C. Newman, "Date and Time on the Internet: Timestamps", RFC 3339, DOI 10.17487/RFC3339, July 2002, <https://www.rfc-editor.org/rfc/rfc3339>.

[GoogleCTLogs] Google, "Certificate Transparency Logs trusted by Google", 5 June 2023, <https://source.chromium.org/chromium/chromium/src/+/main:components/certificate_transparency/data/log_list.json>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/rfc/rfc2119>.

[RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List

(CRL) Profile", RFC 5280, DOI 10.17487/RFC5280, May 2008, <https://www.rfc-editor.org/rfc/rfc5280>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/rfc/rfc8174>.

[TLS13] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <https://www.rfc-editor.org/rfc/rfc8446>.

[TLSCertCompress] Ghedini, A. and V. Vasiliev, "TLS Certificate Compression", RFC 8879, DOI 10.17487/RFC8879, December 2020, <https://www.rfc-editor.org/rfc/rfc8879>.

[ZSTD] Collet, Y. and M. Kucherawy, Ed., "Zstandard Compression and the 'application/zstd' Media Type", RFC 8878, DOI 10.17487/RFC8878, February 2021, <https://www.rfc-editor.org/rfc/rfc8878>.

9.2. Informative References

[CCADB] Mozilla, Microsoft, Google, Apple, and Cisco, "Common CA Database", 5 June 2023, <https://www.ccadb.org/>.

[ECH] Rescorla, E., Oku, K., Sullivan, N., and C. A. Wood, "TLS Encrypted Client Hello", Work in Progress, Internet-Draft, draft-ietf-tls-esni-17, 9 October 2023, <https://datatracker.ietf.org/doc/html/draft-ietf-tls-esni-17>.

[FastlyStudy] McManus, P., "Does the QUIC handshake require compression to be fast?", 18 May 2020, <https://www.fastly.com/blog/quic-handshake-tls-compression-certificates-extension-study>.

[FingerprintingPost] Team, F. S. R., "The state of TLS fingerprinting What's Working, What Isn't, and What's Next", 20 July 2022, <https://www.fastly.com/blog/the-state-of-tls-fingerprinting-whats-working-what-isnt-and-whats-next>.

[I-D.ietf-cose-cbor-encoded-cert-05] Mattsson, J. P., Selander, G., Raza, S., Höglund, J., and M. Furuhed, "CBOR Encoded X.509 Certificates (C509 Certificates)", Work in Progress, Internet-Draft, draft-ietf-cose-cbor-encoded-cert-05, 10 January 2023, <https://datatracker.ietf.org/doc/html/draft-ietf-cose-cbor-encoded-cert-05>.

[I-D.ietf-tls-ctls-08] Rescorla, E., Barnes, R., Tschofenig, H., and B. M. Schwartz, "Compact TLS 1.3", Work in Progress,

Internet-Draft, draft-ietf-tls-ctls-08, 13 March 2023, <https://datatracker.ietf.org/doc/html/draft-ietf-tls-ctls-08>.

[PQStudy] Westerbaan, B., "Sizing Up Post-Quantum Signatures", 8 November 2021, <https://blog.cloudflare.com/sizing-up-post-quantum-signatures/>.

[QUICStudy] Nawrocki, M., Tehrani, P., Hiesgen, R., Mücke, J., Schmidt, T., and M. Wählisch, "On the interplay between TLS certificates and QUIC performance", ACM, Proceedings of the 18th International Conference on emerging Networking EXperiments and Technologies, DOI 10.1145/3555050.3569123, November 2022, <https://doi.org/10.1145/3555050.3569123>.

[RFC7924] Santesson, S. and H. Tschofenig, "Transport Layer Security (TLS) Cached Information Extension", RFC 7924, DOI 10.17487/RFC7924, July 2016, <https://www.rfc-editor.org/rfc/rfc7924>.

[RFC9000] Iyengar, J., Ed. and M. Thomson, Ed., "QUIC: A UDP-Based Multiplexed and Secure Transport", RFC 9000, DOI 10.17487/RFC9000, May 2021, <https://www.rfc-editor.org/rfc/rfc9000>.

[RFC9191] Sethi, M., Preuß Mattsson, J., and S. Turner, "Handling Large Certificates and Long Certificate Chains in TLS-Based EAP Methods", RFC 9191, DOI 10.17487/RFC9191, February 2022, <https://www.rfc-editor.org/rfc/rfc9191>.

[SCA] Kampanakis, P., Bytheway, C., Westerbaan, B., and M. Thomson, "Suppressing CA Certificates in TLS 1.3", Work in Progress, Internet-Draft, draft-kampanakis-tls-scas-latest-03, 5 January 2023, <https://datatracker.ietf.org/doc/html/draft-kampanakis-tls-scas-latest-03>.

[SCAStudy] Kampanakis, P. and M. Kallitsis, "Faster Post-Quantum TLS Handshakes Without Intermediate CA Certificates", Springer International Publishing, Cyber Security, Cryptology, and Machine Learning pp. 337-355, DOI 10.1007/978-3-031-07689-3_25, ISBN ["9783031076886", "9783031076893"], 2022, <https://doi.org/10.1007/978-3-031-07689-3_25>.

[ZstdImpl] Facebook, "ZStandard (Zstd)", 5 June 2023, <https://github.com/facebook/zstd>.

Appendix A. Acknowledgments

The authors thank Bas Westerbaan, Martin Thomson and Kathleen Wilson for feedback on early versions of this document.

Appendix B. CCADB Churn and Dictionary Negotiation

B.1. CCADB Churn

Typically around 10 or so new root certificates are introduced to the WebPKI each year. The various root programs restrict the lifetimes of these certificates, Microsoft to between 8 and 25 years (3.A.3), Mozilla to between 0 and 14 years (Summary). Chrome has proposed a maximum lifetime of 7 years in the future (Blog Post). Some major CAs have objected to this proposed policy as the root inclusion process currently takes around 3 years from start to finish (Digicert Blog). Similarly, Mozilla requires CAs to apply to renew their roots with at least 2 years notice (Summary).

Typically around 100 to 200 new WebPKI intermediate certificates are issued each year. No WebPKI root program currently limits the lifetime of intermediate certificates, but they are in practice capped by the lifetime of their parent root certificate. The vast majority of these certificates are issued with 10 year lifespans. A small but notable fraction (<10%) are issued with 2 or 3 year lifetimes. Chrome's Root Program has proposed that Intermediate Certificates be limited to 3 years in the future (Update). However, the motivation for this requirement is unclear. Unlike root certificates, intermediate certificates are only required to be disclosed with a month's notice to the CCADB (Mozilla Root Program Section 5.3.2, Chrome Policy).

B.2. Dictionary Negotiation

This draft is currently written with a view to being adopted as a particular TLS Certificate Compression Scheme. However, this means that each dictionary used in the wild must have an assigned code point. A new dictionary would likely need to be issued no more than yearly. However, negotiating the dictionary used would avoid the overhead of minting a new draft and code point. A sketch for how dictionary negotiation might work is below.

This draft would instead define a new extension, which would define TLS Certificate Compression with ZStandard Dictionaries. Dictionaries would be identified by an IANA-assigned identifier of two bytes, with a further two bytes for the major version and two more for the minor version. Adding new certificates to a dictionary listing would require a minor version bump. Removing certificates or changing the pass 2 dictionary would require a major version bump.

```
struct {
  uint16 identifier;
  uint16 major_version;
  uint16 minor_version;
} DictionaryId

struct {
  DictionaryId supported_dictionaries<6..2^16-1>
} ZStdSharedDictXtn
```

The client lists their known dictionaries in an extension in the ClientHello. The client need only retain and advertise the highest known minor version for any major version of a dictionary they are willing to offer. The server may select any dictionary it has a copy of with matching identifier, matching major version number and a minor version number not greater than the client's minor version number.

The expectation would be that new minor versions would be issued monthly or quarterly, with new major versions only every year or multiple years. This reflects the relative rates of when certificates are added or removed to the CCADB listing. This means in practice clients would likely offer a single dictionary containing their latest known version. Servers would only need to update their dictionaries yearly when a new major version is produced.

Author's Address

Dennis Jackson
Mozilla

Email: ietf@dennis-jackson.uk