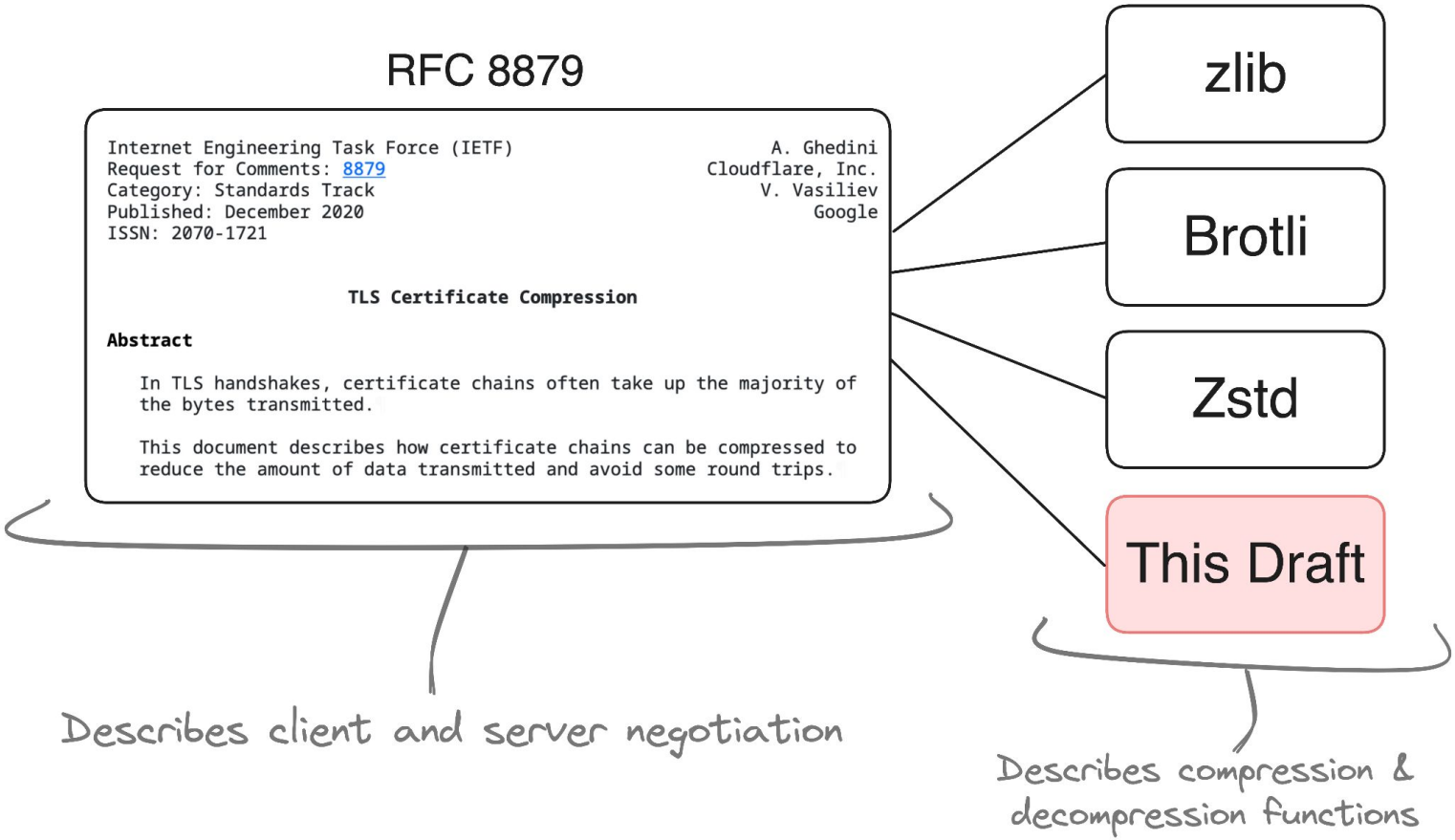
Abridged Certs

draft-jackson-tls-cert-abridge

Abridged Certificate Compression

- A new TLS certificate compression scheme for the WebPKI
- Enables migration to post-quantum **CA Certificates** without size penalties
- Simple and self-contained. Suitable for early experimentation.
- Several details and optimisations still to be worked out.


RFC 8879
Internet Engineering Task Force (IETF)
Request for Comments: 8879
Category: Standards Track
Published: December 2020
ISSN: 2070-1721
A. Ghedini
Cloudflare, Inc.
V. Vasiliev
Google
TLS Certificate Compression
Abstract
In TLS handshakes, certificate chains often take up the majority of the bytes transmitted.
This document describes how certificate chains can be compressed to reduce the amount of data transmitted and avoid some round trips.
zlib
Brotli
Zstd
This Draft
Describes client and server negotiation
Describes compression & decompression functions

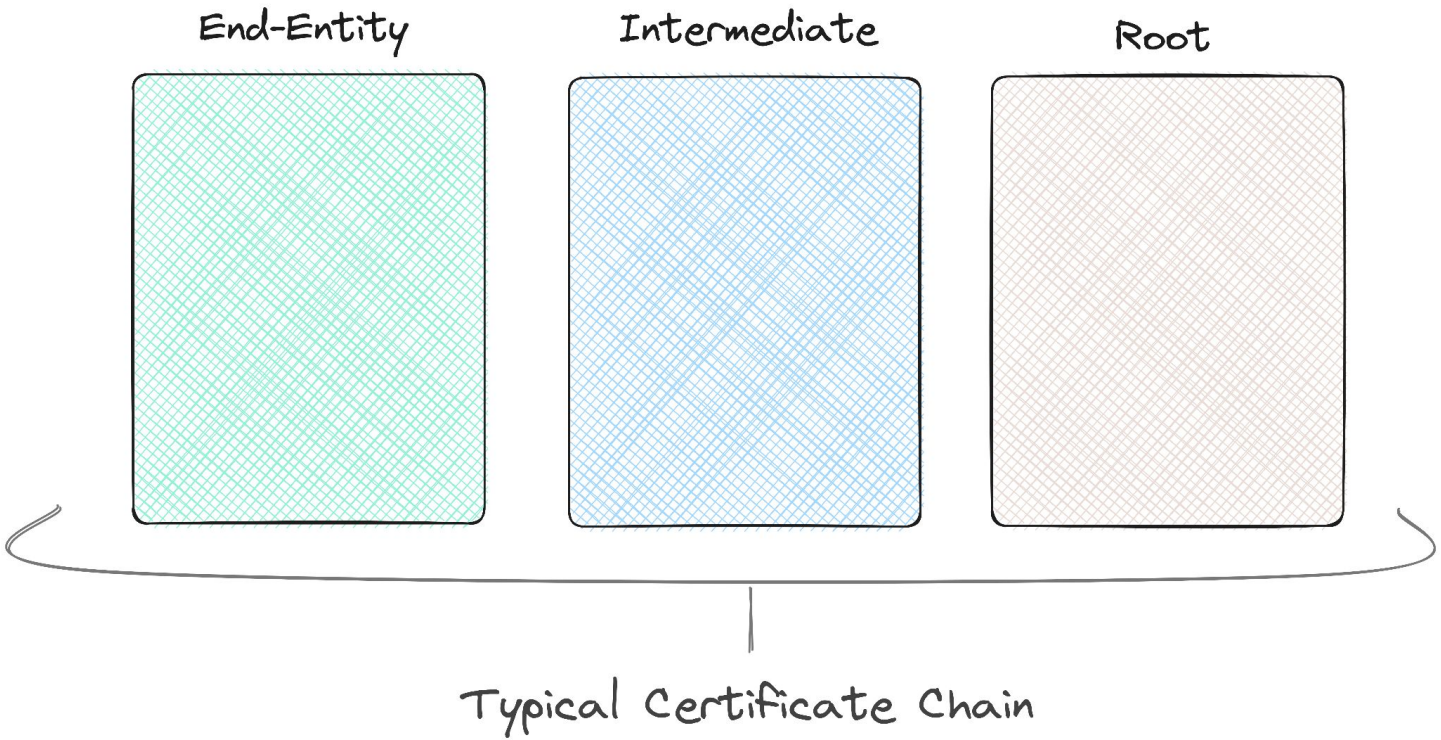
Scheme Walkthrough

Two Passes

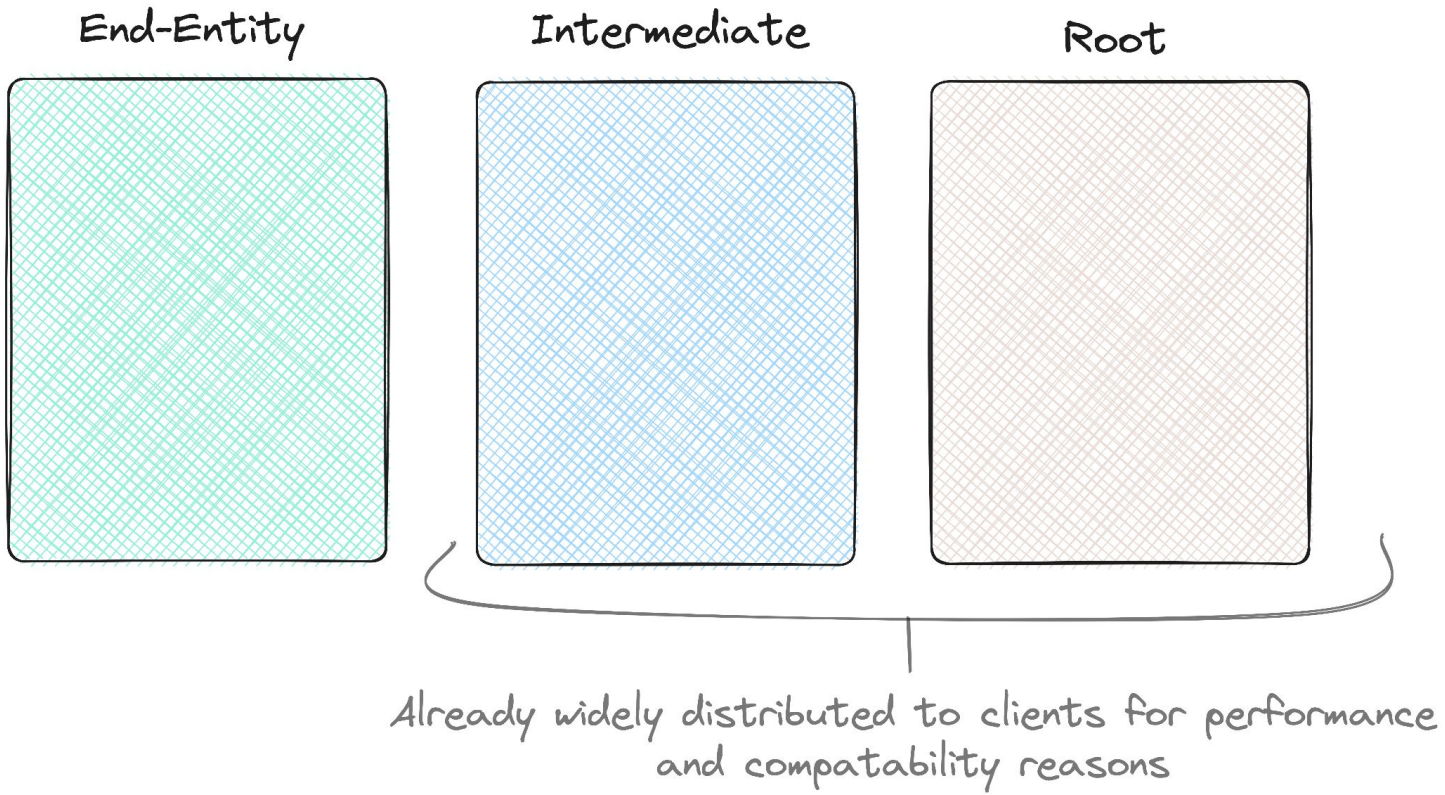
Pass 1 - Compress known intermediate and root certificates

Pass 2 - Compress resulting chain with shared dictionary specific to WebPKI

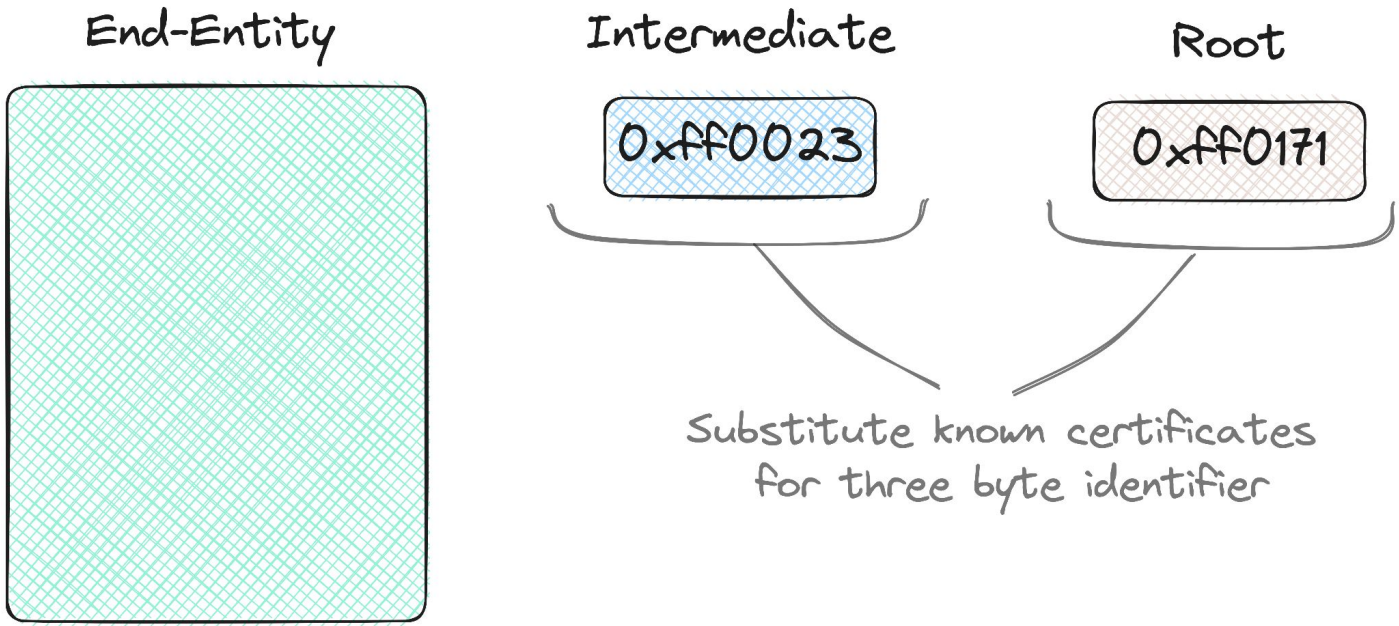
Pass 1



Pass 1


End-Entity
Intermediate
Root
Already widely distributed to clients for performance and compatability reasons

Pass 1

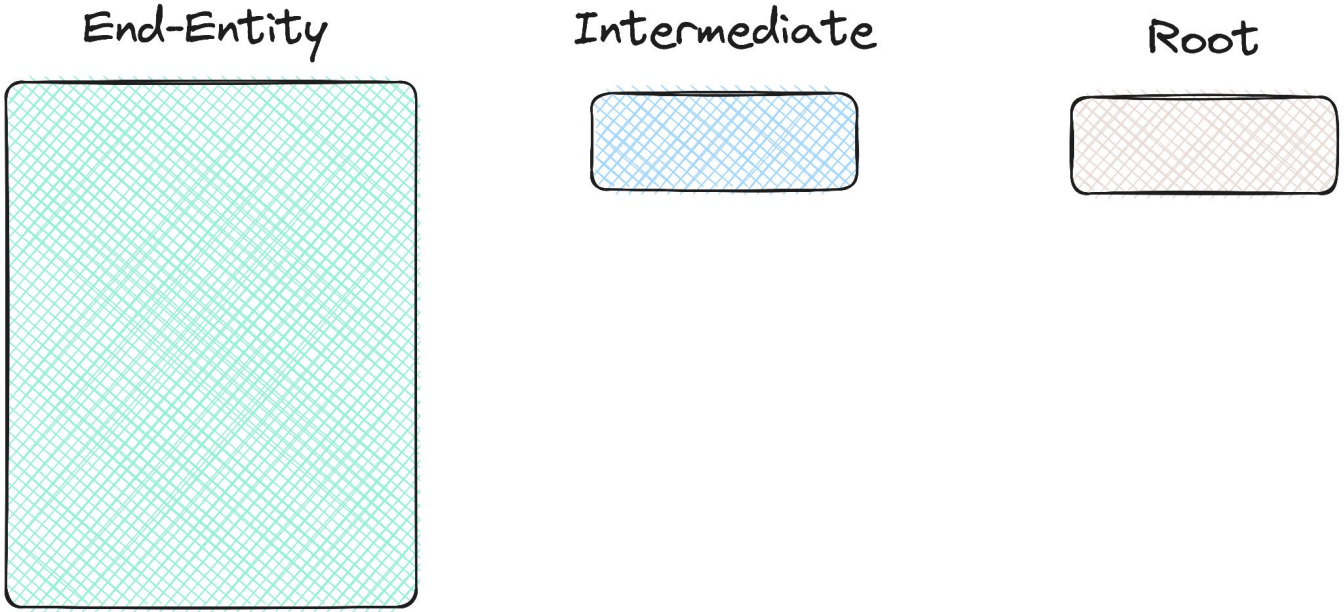

End-Entity
Intermediate
Root
0xff0023
0xff0171
Substitute known certificates
for three byte identifier

Pass 1 - CA Certificate Identifiers

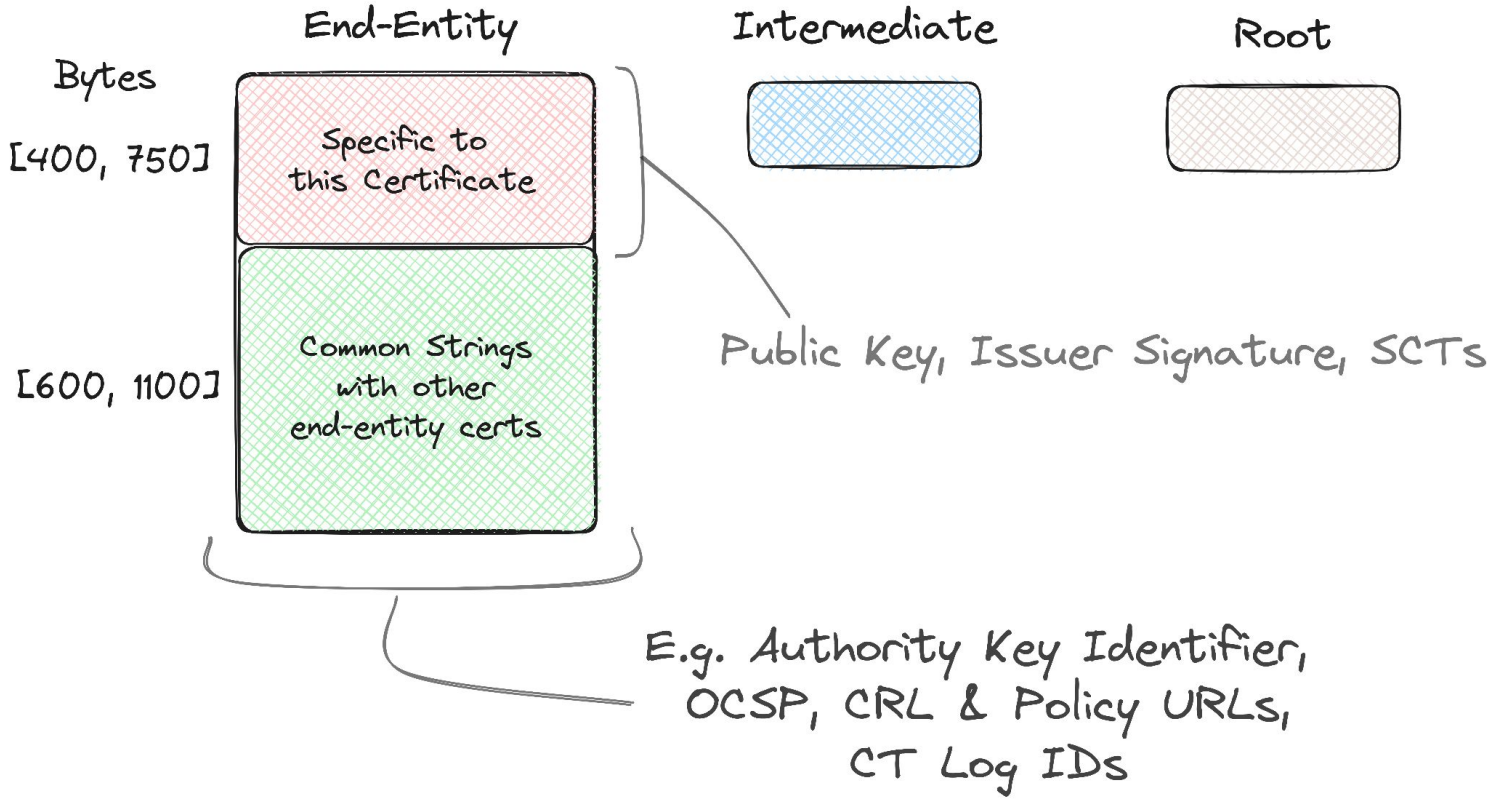
- The **Common CA Database** (CCADB) is operated by Mozilla.
- Apple, Google, Mozilla, Microsoft and others use the CCADB to manage root store changes.

- This draft defines how to select and order the relevant CCADB certs.
- Total of ~2000 certificates which use 3 MB of space. However:
  - Browsers are already shipped with all trusted intermediate and root certificates.
  - Servers don’t need a full certificate, only the hash, in order to offer compression. (0.06 MB)

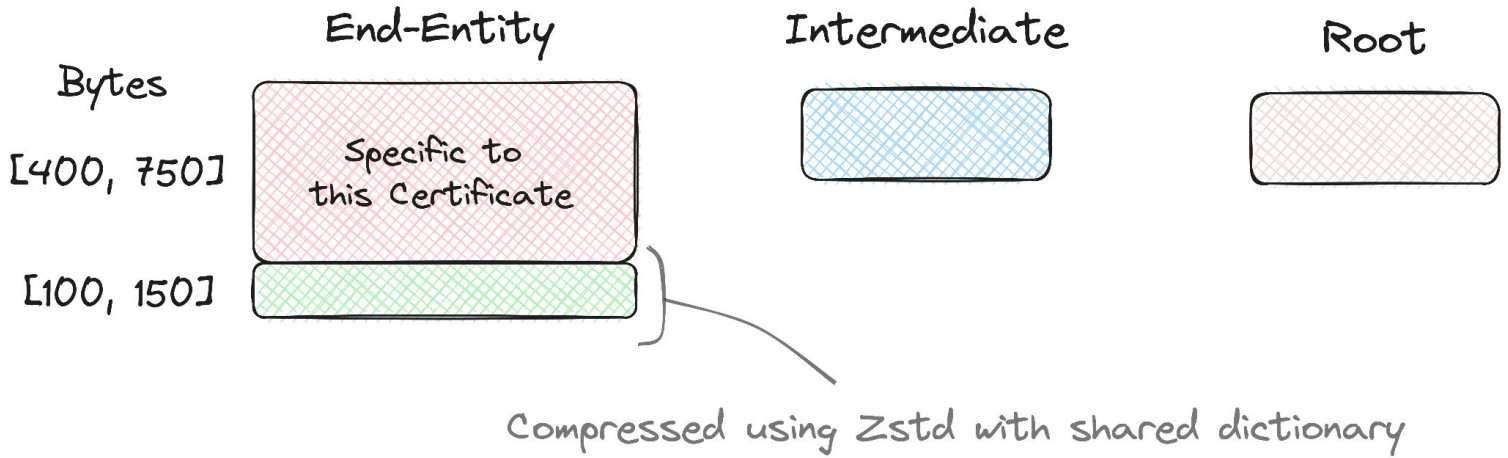
Pass 2



Pass 2


End-Entity
Intermediate
Root
Bytes
[400, 750]
Specific to this Certificate
[600, 1100]
Common Strings with other end-entity certs
Public Key, Issuer Signature, SCTs
E.g. Authority Key Identifier, OCSP, CRL & Policy URLs, CT Log IDs

Pass 2


End-Entity
Intermediate
Root
Bytes
[400, 750]
Specific to this Certificate
[100, 150]
Compressed using Zstd with shared dictionary

Dictionary Contents - Pass 2

- Around 100 KB in total, built by extracting common extensions from CT logs. Roughly 500B per CA.

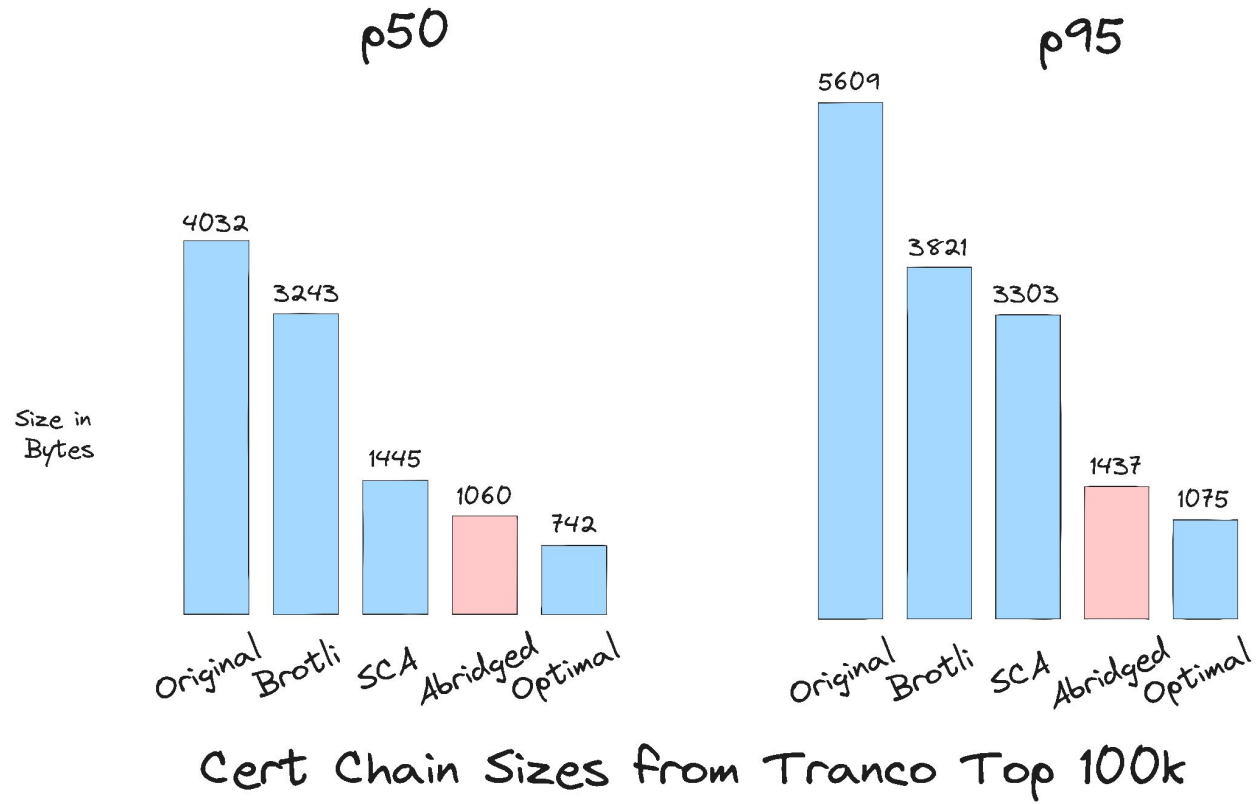
- In the longer term, the CCADB have offered to a host a form where CAs can submit binary data for use with compressing their certs (up to a quota).

- Equitable - CAs contribute to this dictionary equally.
  - Tradeoff: Could use smaller dictionary (e.g. 3KB) with little performance impact but would unfairly benefit popular CAs

Dictionary Versioning

- These dictionaries need to be updated periodically, **likely annually**.
- Adding new certificates to the various root stores typically takes around a year. We can add applicant CAs to dictionary ahead of trust decisions.
- So we can assign **one TLS Cert Compression codepoint to each annual version**. Dictionaries shipped as part of the TLS Library.
- If root stores start to iterate faster, appendix sketches version negotiation.
- No dynamic state or server-side fetches. Avoids OCSP-stapling pitfall.

Evaluation


p50
p95
Size in Bytes
4032
3243
1445
1060
742
5609
3821
3303
1437
1075
Original
Brotli
SCA
Abridged
Optimal
Cert Chain Sizes from Tranco Top 100k

Deployability

- This draft doesn’t introduce any new negotiation or sources of error. No need for retry mechanisms.
- Suitable for deployment as a default in TLS libraries, doesn’t require operator oversight.
- Updates shipped yearly as part of regular software releases. Only binary data would change so easy to backport.
- Equitable for CAs and Websites.

Next Steps & Open Issues

- Integrate feedback from mailing list
- Use CT logs to benchmark different versioning intervals for the dictionary.
- Experimental deployment to measure latency improvements with classical chains.

Discussions from the mailing list:

- Alternative pass 2 dictionary formats? (e.g. multiple small dictionaries)
- Is Pass 2 delivering enough savings to be worthwhile?
- Alternative versioning strategies

Thank Yous

Everyone that gave feedback on the mailing lists and in particular:

- Panos Kampanakis
- Bas Westerbaan
- Kathleen Wilson
- Martin Thomson
- Eric Rescorla

1. Thoughts?

2. WG interest in adopting this draft?

Implementation

```
knownCertificates = [
  0x_00_01 : "TWFueSBo ... ",
  0x_00_02 : "2h0IHdvc ... ",
  ...
]

eeDict = b"bGlnaHQgd29yay ... "

def compress(certChain):
  p1 = [knownCertificates.get(c, default=c) for c in certChain]
  p2 = zstd.compress(b"".join(p1), dict=eeDict, compression_level=MAX)
  return p2
```

- Simple!
- Used with existing TLS library hooks: OpenSSL, BoringSSL, soon NSS
- Tune your own parameters for compression
- Decompression always fast