

TLS
Internet-Draft
Intended status: Standards Track
Expires: May 23, 2020

A. Ghedini
Cloudflare, Inc.
V. Vasiliev
Google
November 20, 2019

TLS Certificate Compression
draft-ietf-tls-certificate-compression-06

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.

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 May 23, 2020.

Copyright Notice

Copyright (c) 2019 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 Simplified BSD License text as described in [Section 4.e](#) of

Internet-Draft

TLS Certificate Compression

November 2019

the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1.	Introduction	2
2.	Notational Conventions	2
3.	Negotiating Certificate Compression	2
4.	Compressed Certificate Message	3
5.	Security Considerations	4
6.	Middlebox Compatibility	5
7.	IANA Considerations	5
7.1.	Update of the TLS ExtensionType Registry	5
7.2.	Update of the TLS HandshakeType Registry	5
7.3.	Registry for Compression Algorithms	5
8.	Normative References	6
Appendix A.	Acknowledgements	7
	Authors' Addresses	7

[1.](#) Introduction

In order to reduce latency and improve performance it can be useful to reduce the amount of data exchanged during a TLS handshake.

[RFC7924] describes a mechanism that allows a client and a server to avoid transmitting certificates already shared in an earlier handshake, but it doesn't help when the client connects to a server for the first time and doesn't already have knowledge of the server's certificate chain.

This document describes a mechanism that would allow certificates to be compressed during all handshakes.

[2.](#) Notational Conventions

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.

[3.](#) Negotiating Certificate Compression

This extension is only supported with TLS 1.3 and newer; if TLS 1.2 or earlier is negotiated, the peers MUST ignore this extension.

This document defines a new extension type (`compress_certificate(27)`), which can be used to signal the supported

compression formats for the Certificate message to the peer. Whenever it is sent by the client as a ClientHello message extension ([\[RFC8446\], Section 4.1.2](#)), it indicates the support for compressed server certificates. Whenever it is sent by the server as a CertificateRequest extension ([\[RFC8446\], Section 4.3.2](#)), it indicates the support for compressed client certificates.

By sending a `compress_certificate` extension, the sender indicates to the peer the certificate compression algorithms it is willing to use for decompression. The "extension_data" field of this extension SHALL contain a `CertificateCompressionAlgorithms` value:

```
enum {
    zlib(1),
    brotli(2),
    zstd(3),
    (65535)
} CertificateCompressionAlgorithm;

struct {
    CertificateCompressionAlgorithm algorithms<2..2^8-2>;
} CertificateCompressionAlgorithms;
```

The `compress_certificate` extension is a unidirectional indication; no corresponding response extension is needed.

[4.](#) Compressed Certificate Message

If the peer has indicated that it supports compression, server and client MAY compress their corresponding Certificate messages and send them in the form of the `CompressedCertificate` message (replacing the Certificate message).

The `CompressedCertificate` message is formed as follows:

```
struct {
```

```
CertificateCompressionAlgorithm algorithm;  
uint24 uncompressed_length;  
opaque compressed_certificate_message<1..2^24-1>;  
} CompressedCertificate;
```

algorithm The algorithm used to compress the certificate. The algorithm MUST be one of the algorithms listed in the peer's compress_certificate extension.

uncompressed_length The length of the Certificate message once it is uncompressed. If after decompression the specified length does not match the actual length, the party receiving the invalid

message MUST abort the connection with the "bad_certificate" alert. The presence of this field allows the receiver to pre-allocate the buffer for the uncompressed Certificate message and to enforce limits on the message size before performing decompression.

compressed_certificate_message The result of applying the indicated compression algorithm to the encoded Certificate message that would have been sent if certificate compression was not in use. The compression algorithm defines how the bytes in the compressed_certificate_message field are converted into the Certificate message.

If the specified compression algorithm is zlib, then the Certificate message MUST be compressed with the ZLIB compression algorithm, as defined in [\[RFC1950\]](#). If the specified compression algorithm is brotli, the Certificate message MUST be compressed with the Brotli compression algorithm as defined in [\[RFC7932\]](#). If the specified compression algorithm is zstd, the Certificate message MUST be compressed with the Zstandard compression algorithm as defined in [\[I-D.kucherawy-rfc8478bis\]](#).

It is possible to define a certificate compression algorithm that uses a pre-shared dictionary to achieve higher compression ratio. This document does not define any such algorithms, but additional codepoints may be allocated for such use per the policy in [Section 7.3](#).

If the received CompressedCertificate message cannot be decompressed,

the connection MUST be terminated with the "bad_certificate" alert.

If the format of the Certificate message is altered using the `server_certificate_type` or `client_certificate_type` extensions [[RFC7250](#)], the resulting altered message is compressed instead.

5. Security Considerations

After decompression, the Certificate message MUST be processed as if it were encoded without being compressed. This way, the parsing and the verification have the same security properties as they would have in TLS normally.

In order for certificate compression to function correctly, the underlying compression algorithm MUST output the same data that was provided as input by the peer.

Since certificate chains are typically presented on a per-server name or per-user basis, a malicious application does not have control over

any individual fragments in the Certificate message, meaning that they cannot leak information about the certificate by modifying the plaintext.

Implementations SHOULD bound the memory usage when decompressing the `CompressedCertificate` message.

Implementations MUST limit the size of the resulting decompressed chain to the specified uncompressed length, and they MUST abort the connection if the size of the output of the decompression function exceeds that limit. TLS framing imposes 16777216 byte limit on the certificate message size, and the implementations MAY impose a limit that is lower than that; in both cases, they MUST apply the same limit as if no compression were used.

6. Middlebox Compatibility

It's been observed that a significant number of middleboxes intercept and try to validate the Certificate message exchanged during a TLS handshake. This means that middleboxes that don't understand the `CompressedCertificate` message might misbehave and drop connections that adopt certificate compression. Because of that, the extension

is only supported in the versions of TLS where the certificate message is encrypted in a way that prevents middleboxes from intercepting it, that is, TLS version 1.3 [[RFC8446](#)] and higher.

[7.](#) IANA Considerations

[7.1.](#) Update of the TLS ExtensionType Registry

Create an entry, `compress_certificate(27)`, in the existing registry for ExtensionType (defined in [[RFC8446](#)]), with "TLS 1.3" column values being set to "CH, CR", and "Recommended" column being set to "Yes".

[7.2.](#) Update of the TLS HandshakeType Registry

Create an entry, `compressed_certificate(25)`, in the existing registry for HandshakeType (defined in [[RFC8446](#)]).

[7.3.](#) Registry for Compression Algorithms

This document establishes a registry of compression algorithms supported for compressing the Certificate message, titled "Certificate Compression Algorithm IDs", under the existing "Transport Layer Security (TLS) Extensions" heading.

The entries in the registry are:

+-----+-----+	
Algorithm Number	Description
+-----+-----+	
0	Reserved
1	zlib
2	brotnli
3	zstd
16384 to 65535	Reserved for Experimental Use
+-----+-----+	

The values in this registry shall be allocated under "IETF Review"

policy for values strictly smaller than 256, under "Specification Required" policy for values 256-16383, and under "Experimental Use" otherwise (see [RFC8126] for the definition of relevant policies). Experimental Use extensions can be used both on private networks and over the open Internet.

The procedures for requesting values in the Specification Required space are specified in [RFC8447].

8. Normative References

- [I-D.kuchera-wy-rfc8478bis]
Collet, Y. and M. Kuchera-wy, "Zstandard Compression and the application/zstd Media Type", [draft-kuchera-wy-rfc8478bis-00](#) (work in progress), October 2019.
- [RFC1950] Deutsch, P. and J-L. Gailly, "ZLIB Compressed Data Format Specification version 3.3", [RFC 1950](#), DOI 10.17487/RFC1950, May 1996, <<https://www.rfc-editor.org/info/rfc1950>>.
- [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/info/rfc2119>>.
- [RFC7250] Wouters, P., Ed., Tschofenig, H., Ed., Gilmore, J., Weiler, S., and T. Kivinen, "Using Raw Public Keys in Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)", [RFC 7250](#), DOI 10.17487/RFC7250, June 2014, <<https://www.rfc-editor.org/info/rfc7250>>.

- [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/info/rfc7924>>.
- [RFC7932] Alakuujala, J. and Z. Szabadka, "Brotli Compressed Data Format", [RFC 7932](#), DOI 10.17487/RFC7932, July 2016, <<https://www.rfc-editor.org/info/rfc7932>>.

- [RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", [BCP 26](#), [RFC 8126](#), DOI 10.17487/RFC8126, June 2017, <<https://www.rfc-editor.org/info/rfc8126>>.
- [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/info/rfc8174>>.
- [RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", [RFC 8446](#), DOI 10.17487/RFC8446, August 2018, <<https://www.rfc-editor.org/info/rfc8446>>.
- [RFC8447] Salowey, J. and S. Turner, "IANA Registry Updates for TLS and DTLS", [RFC 8447](#), DOI 10.17487/RFC8447, August 2018, <<https://www.rfc-editor.org/info/rfc8447>>.

[Appendix A](#). Acknowledgements

Certificate compression was originally introduced in the QUIC Crypto protocol, designed by Adam Langley and Wan-Teh Chang.

This document has benefited from contributions and suggestions from David Benjamin, Ryan Hamilton, Ilari Liusvaara, Piotr Sikora, Ian Swett, Martin Thomson, Sean Turner and many others.

Authors' Addresses

Alessandro Ghedini
Cloudflare, Inc.

Email: alessandro@cloudflare.com

Victor Vasiliev
Google

Email: vasilvv@google.com