TLS A. Ghedini
Internet-Draft Cloudflare, Inc.
Intended status: Standards Track V. Vasiliev
Expires: October 7, 2019 Google
April 05, 2019

TLS Certificate Compression
draft-ietf-tls-certificate-compression-05

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

Copyright Notice

Table of Contents

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 full 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;
```

There is no ServerHello extension that the server is required to echo back.

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 compressed body of the Certificate message, in the same format as it would normally be expressed in. 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 [RFC8478].

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.

If the received CompressedCertificate message cannot be decompressed, the connection MUST be torn down 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 be deterministic and it 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, the attacker 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.

The implementations SHOULD bound the memory usage when decompressing the CompressedCertificate message.

The implementations MUST limit the size of the resulting decompressed chain to the specified uncompressed length, and they MUST abort the connection if the size 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	brotli
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

[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] Alakuijala, 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>.

[RFC8478] Collet, Y. and M. Kucherawy, Ed., "Zstandard Compression and the application/zstd Media Type", RFC 8478, DOI 10.17487/RFC8478, October 2018, <https://www.rfc-editor.org/info/rfc8478>.

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