

Network Working Group  
Internet-Draft  
Intended status: Standards Track  
Expires: September 2, 2017

A. Ghedini  
Cloudflare, Inc.  
V. Vasiliev  
Google  
March 01, 2017

Transport Layer Security (TLS) Certificate Compression  
draft-ghedini-tls-certificate-compression-00

## Abstract

In Transport Layer Security (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 <http://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 September 2, 2017.

## Copyright Notice

Copyright (c) 2017 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 (<http://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

March 2017

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

## Table of Contents

<a href="#">1.</a>	Introduction . . . . .	<a href="#">2</a>
<a href="#">2.</a>	Notational Conventions . . . . .	<a href="#">2</a>
<a href="#">3.</a>	Negotiating Certificate Compression . . . . .	<a href="#">2</a>
<a href="#">4.</a>	Server Certificate Message . . . . .	<a href="#">3</a>
<a href="#">5.</a>	Security Considerations . . . . .	<a href="#">4</a>
<a href="#">6.</a>	IANA Considerations . . . . .	<a href="#">4</a>
<a href="#">6.1.</a>	Update of the TLS ExtensionType Registry . . . . .	<a href="#">4</a>
<a href="#">6.2.</a>	Registry for Compression Algorithms . . . . .	<a href="#">5</a>
<a href="#">7.</a>	Normative References . . . . .	<a href="#">5</a>
	Authors' Addresses . . . . .	<a href="#">6</a>

## [1.](#) Introduction

In order to reduce latency and improve performance it can be useful to reduce the amount of data exchanged during a Transport Layer Security (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 server certificates to be compressed during full handshakes.

## [2.](#) Notational Conventions

The words "MUST", "MUST NOT", "SHALL", "SHOULD", and "MAY" are used in this document. It's not shouting; when they are capitalized, they have the special meaning defined in [[RFC2119](#)].

## [3.](#) Negotiating Certificate Compression

This document defines a new extension type (`compress_server_certificates(TBD)`), which is used by the client and the server to negotiate the use of compression for the server certificate chain, as well as the choice of the compression

algorithm.

By sending the `compress_server_certificates` message, the client indicates to the server the certificate compression algorithms it

supports. The "extension\_data" field of this extension in the ClientHello SHALL contain a `CertificateCompressionAlgorithms` value:

```
enum {
    zlib(0),
    brotli(1),
    (255)
} CertificateCompressionAlgorithm;

struct {
    CertificateCompressionAlgorithm algorithms<1..2^8>;
} CertificateCompressionAlgorithms;
```

If the server supports any of the algorithms offered in the ClientHello, it MAY respond with an extension indicating which compression algorithm it chose. In that case, the `extension_data` SHALL be a `CertificateCompressionAlgorithm` value corresponding to the chosen algorithm. If the server has chosen to not use any compression, it MUST NOT send the `compress_server_certificates` extension.

#### [4.](#) Server Certificate Message

If the server picks a compression algorithm and sends it in the ServerHello, the format of the Certificate message is altered as follows:

```
struct {
    uint24 uncompressed_length;
    opaque compressed_certificate_message<1..2^24-1>;
} Certificate;
```

`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 client MUST abort the connection with the "bad\_certificate" alert.

`compressed_certificate_message` The compressed body of the Certificate message, in the same format as the server would normally express it. The compression algorithm defines how the bytes in the `compressed_certificate_message` 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 client cannot decompress the received Certificate message from the server, it MUST tear down the connection with the "bad\_certificate" alert.

The extension only affects the Certificate message from the server. It does not change the format of the Certificate message sent by the client.

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

If the server chooses to use the `cached_info` extension [[RFC7924](#)] to replace the Certificate message with a hash, it MUST NOT send the `compress_server_certificates` extension.

## [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.

Since certificate chains are typically presented on a per-server name 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 Certificate 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. Implementations MAY impose a lower limit on the chain size in addition to the 16777216 byte limit imposed by TLS framing, in which case they MUST apply the same limit to the uncompressed chain before starting to decompress it.

## [6.](#) IANA Considerations

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

Create an entry, `compress_server_certificates(TBD)`, in the existing registry for ExtensionType (defined in [[RFC5246](#)]).

### [6.2.](#) 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	zlib
1	brotli
224 to 255	Reserved for Private Use

The values in this registry shall be allocated under "IETF Review" policy for values strictly smaller than 64, and under "Specification Required" policy otherwise (see [[RFC5226](#)] for the definition of

relevant policies).

## 7. 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, <<http://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, <<http://www.rfc-editor.org/info/rfc2119>>.
- [RFC4366] Blake-Wilson, S., Nystrom, M., Hopwood, D., Mikkelsen, J., and T. Wright, "Transport Layer Security (TLS) Extensions", [RFC 4366](#), DOI 10.17487/RFC4366, April 2006, <<http://www.rfc-editor.org/info/rfc4366>>.
- [RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", [BCP 26](#), [RFC 5226](#), DOI 10.17487/RFC5226, May 2008, <<http://www.rfc-editor.org/info/rfc5226>>.

- [RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", [RFC 5246](#), DOI 10.17487/RFC5246, August 2008, <<http://www.rfc-editor.org/info/rfc5246>>.
- [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, <<http://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, <<http://www.rfc-editor.org/info/rfc7924>>.

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

#### Authors' Addresses

Alessandro Ghedini  
Cloudflare, Inc.

Email: [alessandro@cloudflare.com](mailto:alessandro@cloudflare.com)

Victor Vasiliev  
Google

Email: [vasilvv@google.com](mailto:vasilvv@google.com)