

Hypertext Transfer Protocol  
Internet-Draft  
Intended status: Standards Track  
Expires: September 29, 2017

M. Bishop  
Microsoft  
N. Sullivan  
Cloudflare  
M. Thomson  
Mozilla  
March 28, 2017

**Secondary Certificate Authentication in HTTP/2**  
**draft-bishop-httpbis-http2-additional-certs-03**

Abstract

TLS provides fundamental mutual authentication services for HTTP, supporting up to one server certificate and up to one client certificate associated to the session to prove client and server identities as necessary. This draft provides mechanisms for providing additional such certificates at the HTTP layer when these constraints are not sufficient.

Many HTTP servers host content from several origins. HTTP/2 [[RFC7540](#)] permits clients to reuse an existing HTTP connection to a server provided that the secondary origin is also in the certificate provided during the TLS [[I-D.ietf-tls-tls13](#)] handshake.

In many cases, servers will wish to maintain separate certificates for different origins but still desire the benefits of a shared HTTP connection. Similarly, servers may require clients to present authentication, but have different requirements based on the content the client is attempting to access.

This document describes how TLS exported authenticators [[I-D.sullivan-tls-exported-authenticator](#)] can be used to provide proof of ownership of additional certificates to the HTTP layer to support both scenarios.

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 29, 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 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="#">3</a>
<a href="#">1.1.</a>	Server Certificate Authentication . . . . .	<a href="#">3</a>
<a href="#">1.2.</a>	Client Certificate Authentication . . . . .	<a href="#">4</a>
<a href="#">1.2.1.</a>	HTTP/1.1 using TLS 1.2 and previous . . . . .	<a href="#">4</a>
<a href="#">1.2.2.</a>	HTTP/1.1 using TLS 1.3 . . . . .	<a href="#">5</a>
<a href="#">1.2.3.</a>	HTTP/2 . . . . .	<a href="#">6</a>
<a href="#">1.3.</a>	HTTP-Layer Certificate Authentication . . . . .	<a href="#">6</a>
<a href="#">1.4.</a>	Terminology . . . . .	<a href="#">7</a>
<a href="#">2.</a>	Discovering Additional Certificates at the HTTP/2 Layer . . .	<a href="#">8</a>
<a href="#">2.1.</a>	Indicating support for HTTP-layer certificate authentication . . . . .	<a href="#">8</a>
<a href="#">2.2.</a>	Making certificates or requests available . . . . .	<a href="#">8</a>
<a href="#">2.3.</a>	Requiring certificate authentication . . . . .	<a href="#">9</a>
<a href="#">3.</a>	Certificates Frames for HTTP/2 . . . . .	<a href="#">11</a>
<a href="#">3.1.</a>	The CERTIFICATE_NEEDED frame . . . . .	<a href="#">11</a>
<a href="#">3.2.</a>	The USE_CERTIFICATE Frame . . . . .	<a href="#">12</a>
<a href="#">3.3.</a>	The CERTIFICATE_REQUEST Frame . . . . .	<a href="#">13</a>
<a href="#">3.4.</a>	The CERTIFICATE Frame . . . . .	<a href="#">14</a>
<a href="#">4.</a>	Indicating failures during HTTP-Layer Certificate Authentication . . . . .	<a href="#">15</a>
<a href="#">5.</a>	Security Considerations . . . . .	<a href="#">16</a>
<a href="#">5.1.</a>	Impersonation . . . . .	<a href="#">16</a>
<a href="#">5.2.</a>	Fingerprinting . . . . .	<a href="#">17</a>
<a href="#">5.3.</a>	Denial of Service . . . . .	<a href="#">17</a>



5.4.	Confusion About State . . . . .	18
6.	IANA Considerations . . . . .	18
6.1.	HTTP/2 SETTINGS_HTTP_CERT_AUTH Setting . . . . .	18
6.2.	New HTTP/2 Frames . . . . .	18
6.3.	New HTTP/2 Error Codes . . . . .	19
7.	Acknowledgements . . . . .	19
8.	References . . . . .	19
8.1.	Normative References . . . . .	19
8.2.	Informative References . . . . .	21
	Authors' Addresses . . . . .	21

## **1. Introduction**

HTTP clients need to know that the content they receive on a connection comes from the origin that they intended to retrieve in from. The traditional form of server authentication in HTTP has been in the form of X.509 certificates provided during the TLS [RFC5246](#) [[I-D.ietf-tls-tls13](#)] handshake.

Many existing HTTP [RFC7230](#) servers also have authentication requirements for the resources they serve. Of the bountiful authentication options available for authenticating HTTP requests, client certificates present a unique challenge for resource-specific authentication requirements because of the interaction with the underlying TLS layer.

TLS 1.2 [RFC5246](#) supports one server and one client certificate on a connection. These certificates may contain multiple identities, but only one certificate may be provided.

### **1.1. Server Certificate Authentication**

[Section 9.1.1 of RFC7540](#) describes how connections may be used to make requests from multiple origins as long as the server is authoritative for both. A server is considered authoritative for an origin if DNS resolves the origin to the IP address of the server and (for TLS) if the certificate presented by the server contains the origin in the Subject Alternative Names field.

[RFC7838] enables a step of abstraction from the DNS resolution. If both hosts have provided an Alternative Service at hostnames which resolve to the IP address of the server, they are considered authoritative just as if DNS resolved the origin itself to that address. However, the server's one TLS certificate is still required to contain the name of each origin in question.

Servers which host many origins often would prefer to have separate certificates for some sets of origins. This may be for ease of



certificate management (the ability to separately revoke or renew them), due to different sources of certificates (a CDN acting on behalf of multiple origins), or other factors which might drive this administrative decision. Clients connecting to such origins cannot currently reuse connections, even if both client and server would prefer to do so.

Because the TLS SNI extension is exchanged in the clear, clients might also prefer to retrieve certificates inside the encrypted context. When this information is sensitive, it might be advantageous to request a general-purpose certificate or anonymous ciphersuite at the TLS layer, while acquiring the "real" certificate in HTTP after the connection is established.

## **1.2. Client Certificate Authentication**

For servers that wish to use client certificates to authenticate users, they might request client authentication during or immediately after the TLS handshake. However, if not all users or resources need certificate-based authentication, a request for a certificate has the unfortunate consequence of triggering the client to seek a certificate, possibly requiring user interaction, network traffic, or other time-consuming activities. During this time, the connection is stalled in many implementations. Such a request can result in a poor experience, particularly when sent to a client that does not expect the request.

The TLS 1.3 CertificateRequest can be used by servers to give clients hints about which certificate to offer. Servers that rely on certificate-based authentication might request different certificates for different resources. Such a server cannot use contextual information about the resource to construct an appropriate TLS CertificateRequest message during the initial handshake.

Consequently, client certificates are requested at connection establishment time only in cases where all clients are expected or required to have a single certificate that is used for all resources. Many other uses for client certificates are reactive, that is, certificates are requested in response to the client making a request.

### **1.2.1. HTTP/1.1 using TLS 1.2 and previous**

In HTTP/1.1, a server that relies on client authentication for a subset of users or resources does not request a certificate when the connection is established. Instead, it only requests a client certificate when a request is made to a resource that requires a certificate. TLS 1.2 [[RFC5246](#)] accomodates this by permitting the



server to request a new TLS handshake, in which the server will request the client's certificate.

Figure 1 shows the server initiating a TLS-layer renegotiation in response to receiving an HTTP/1.1 request to a protected resource.

Client	Server
-- (HTTP) GET /protected	-----> *1
<-----	(TLS) HelloRequest -- *2
-- (TLS) ClientHello	----->
<-----	(TLS) ServerHello, ... --
<-----	(TLS) CertificateRequest -- *3
-- (TLS) ..., Certificate	-----> *4
-- (TLS) Finished	----->
<-----	(TLS) Finished --
<-----	(HTTP) 200 OK -- *5

Figure 1: HTTP/1.1 Reactive Certificate Authentication with TLS 1.2

In this example, the server receives a request for a protected resource (at \*1 on Figure 1). Upon performing an authorization check, the server determines that the request requires authentication using a client certificate and that no such certificate has been provided.

The server initiates TLS renegotiation by sending a TLS HelloRequest (at \*2). The client then initiates a TLS handshake. Note that some TLS messages are elided from the figure for the sake of brevity.

The critical messages for this example are the server requesting a certificate with a TLS CertificateRequest (\*3); this request might use information about the request or resource. The client then provides a certificate and proof of possession of the private key in Certificate and CertificateVerify messages (\*4).

When the handshake completes, the server performs any authorization checks a second time. With the client certificate available, it then authorizes the request and provides a response (\*5).

### **1.2.2. HTTP/1.1 using TLS 1.3**

TLS 1.3 [[I-D.ietf-tls-tls13](#)] introduces a new client authentication mechanism that allows for clients to authenticate after the handshake has been completed. For the purposes of authenticating an HTTP request, this is functionally equivalent to renegotiation. Figure 2 shows the simpler exchange this enables.





```
Client                                Server
-- (HTTP) GET /protected ----->
<----- (TLS) CertificateRequest --
-- (TLS) Certificate, CertificateVerify,
      Finished ----->
<----- (HTTP) 200 OK --
```

Figure 2: HTTP/1.1 Reactive Certificate Authentication with TLS 1.3

TLS 1.3 does not support renegotiation, instead supporting direct client authentication. In contrast to the TLS 1.2 example, in TLS 1.3, a server can simply request a certificate.

### [1.2.3.](#) HTTP/2

An important part of the HTTP/1.1 exchange is that the client is able to easily identify the request that caused the TLS renegotiation. The client is able to assume that the next unanswered request on the connection is responsible. The HTTP stack in the client is then able to direct the certificate request to the application or component that initiated that request. This ensures that the application has the right contextual information for processing the request.

In HTTP/2, a client can have multiple outstanding requests. Without some sort of correlation information, a client is unable to identify which request caused the server to request a certificate.

Thus, the minimum necessary mechanism to support reactive certificate authentication in HTTP/2 is an identifier that can be used to correlate an HTTP request with a request for a certificate. Since streams are used for individual requests, correlation with a stream is sufficient.

[RFC7540] prohibits renegotiation after any application data has been sent. This completely blocks reactive certificate authentication in HTTP/2 using TLS 1.2. If this restriction were relaxed by an extension or update to HTTP/2, such an identifier could be added to TLS 1.2 by means of an extension to TLS. Unfortunately, many TLS 1.2 implementations do not permit application data to continue during a renegotiation. This is problematic for a multiplexed protocol like HTTP/2.

### [1.3.](#) HTTP-Layer Certificate Authentication

This draft defines HTTP/2 frames to carry the relevant certificate messages, enabling certificate-based authentication of both clients and servers independent of TLS version. This mechanism can be



implemented at the HTTP layer without breaking the existing interface between HTTP and applications above it.

This could be done in a naive manner by replicating the TLS messages as HTTP/2 frames on each stream. However, this would create needless redundancy between streams and require frequent expensive signing operations. Instead, TLS Exported Authenticators [[I-D.sullivan-tls-exported-authenticator](#)] are exchanged on stream zero and the on-stream frames incorporate them by reference as needed.

TLS Exported Authenticators are structured messages that can be exported by either party of a TLS connection and validated by the other party. An authenticator message can be constructed by either the client or the server given an established TLS connection, a certificate, and a corresponding private key. Exported Authenticators use the message structures from section 4.4 of [[I-D.ietf-tls-tls13](#)], but different parameters.

Each Authenticator is computed using a Handshake Context and Finished MAC Key derived from the TLS session. The Handshake Context is identical for both parties of the TLS connection, while the Finished MAC Key is dependent on whether the Authenticator is created by the client or the server.

Successfully verified Authenticators result in certificate chains, with verified possession of the corresponding private key, which can be supplied into a collection of available certificates. Likewise, descriptions of desired certificates can be supplied into these collections. These pre-supplied elements are then available for automatic use (in some situations) or for reference by individual streams.

[Section 2](#) describes how the feature is employed, defining means to detect support in peers ([Section 2.1](#)), make certificates and requests available ([Section 2.2](#)), and indicate when streams are blocked waiting on an appropriate certificate ([Section 2.3](#)). [Section 3](#) defines the required frame types, which parallel the TLS 1.3 message exchange. Finally, [Section 4](#) defines new error types which can be used to notify peers when the exchange has not been successful.

#### **[1.4. Terminology](#)**

[RFC 2119](#) [[RFC2119](#)] defines the terms "MUST", "MUST NOT", "SHOULD" and "MAY".



## **2. Discovering Additional Certificates at the HTTP/2 Layer**

A certificate chain with proof of possession of the private key corresponding to the end-entity certificate is sent as a single "CERTIFICATE" frame (see [Section 3.4](#)) on stream zero. Once the holder of a certificate has sent the chain and proof, this certificate chain is cached by the recipient and available for future use. If the certificate is marked as "AUTOMATIC\_USE", the certificate may be used by the recipient to authorize any current or future request. Otherwise, the recipient requests the required certificate on each stream, but the previously-supplied certificates are available for reference without having to resend them.

Likewise, the details of a request are sent on stream zero and stored by the recipient. These details will be referenced by subsequent "CERTIFICATE\_NEEDED" frames.

Data sent by each peer is correlated by the ID given in each frame. This ID is unrelated to values used by the other peer, even if each uses the same ID in certain cases.

### **2.1. Indicating support for HTTP-layer certificate authentication**

Clients and servers that will accept requests for HTTP-layer certificate authentication indicate this using the HTTP/2 "SETTINGS\_HTTP\_CERT\_AUTH" (0xSETTING-TBD) setting.

The initial value for the "SETTINGS\_HTTP\_CERT\_AUTH" setting is 0, indicating that the peer does not support HTTP-layer certificate authentication. If a peer does support HTTP-layer certificate authentication, the value is 1.

### **2.2. Making certificates or requests available**

When a peer has advertised support for HTTP-layer certificates as in [Section 2.1](#), either party can supply additional certificates into the connection at any time. These certificates then become available for the peer to consider when deciding whether a connection is suitable to transport a particular request.

Available certificates which have the "AUTOMATIC\_USE" flag set MAY be used by the recipient without further notice. This means that clients or servers which predict a certificate will be required could pre-supply the certificate without being asked. Regardless of whether "AUTOMATIC\_USE" is set, these certificates are available for reference by future "USE\_CERTIFICATE" frames.



```

Client                                Server
<----- (stream 0) CERTIFICATE (AU flag) --
...
-- (stream N) GET /from-new-origin ----->
<----- (stream N) 200 OK --

```

Figure 3: Proactive Server Certificate

```

Client                                Server
-- (stream 0) CERTIFICATE (AU flag) ----->
-- (streams 1,3) GET /protected ----->
<----- (streams 1,3) 200 OK --

```

Figure 4: Proactive Client Certificate

Likewise, either party can supply a "CERTIFICATE\_REQUEST" that outlines parameters of a certificate they might request in the future. It is important to note that this does not currently request such a certificate, but makes the contents of the request available for reference by a future "CERTIFICATE\_NEEDED" frame.

### 2.3. Requiring certificate authentication

As defined in [[RFC7540](#)], when a client finds that a https:// origin (or Alternative Service [[RFC7838](#)]) to which it needs to make a request has the same IP address as a server to which it is already connected, it MAY check whether the TLS certificate provided contains the new origin as well, and if so, reuse the connection.

If the TLS certificate does not contain the new origin, but the server has advertised support for HTTP-layer certificates (see [Section 2.1](#)), it MAY send a "CERTIFICATE\_NEEDED" frame on the stream it will use to make the request. (If the request parameters have not already been made available using a "CERTIFICATE\_REQUEST" frame, the client will need to send the "CERTIFICATE\_REQUEST" in order to generate the "CERTIFICATE\_NEEDED" frame.) The stream represents a pending request to that origin which is blocked until a valid certificate is processed.

The request is blocked until the server has responded with a "USE\_CERTIFICATE" frame pointing to a certificate for that origin. If the certificate is already available, the server SHOULD immediately respond with the appropriate "USE\_CERTIFICATE" frame. (If the certificate has not already been transmitted, the server will need to make the certificate available as described in [Section 2.2](#) before completing the exchange.)





If the server does not have the desired certificate or cannot produce a signature compatible with the client's advertised settings, it MUST respond with an empty "USE\_CERTIFICATE" frame. In this case, or if the server has not advertised support for HTTP-layer certificates, the client MUST NOT send any requests for resources in that origin on the current connection.

Client	Server
-- (stream 0) CERTIFICATE_REQUEST ----->	
...	
-- (stream N) CERTIFICATE_NEEDED ----->	
<----- (stream 0) CERTIFICATE --	
<----- (stream N) USE_CERTIFICATE --	
-- (stream N) GET /from-new-origin ----->	
<----- (stream N) 200 OK --	

Figure 5: Client-Requested Certificate

Likewise, on each stream where certificate authentication is required, the server sends a "CERTIFICATE\_NEEDED" frame, which the client answers with a "USE\_CERTIFICATE" frame indicating the certificate to use. If the request parameters or the responding certificate are not already available, they will need to be sent as described in [Section 2.2](#) as part of this exchange.

Client	Server
<----- (stream 0) CERTIFICATE_REQUEST --	
...	
-- (stream N) GET /protected ----->	
<----- (stream N) CERTIFICATE_NEEDED --	
-- (stream 0) CERTIFICATE ----->	
-- (stream N) USE_CERTIFICATE ----->	
<----- (stream N) 200 OK --	

Figure 6: Reactive Certificate Authentication

A server SHOULD provide certificates for an origin before pushing resources from it. If a client receives a "PUSH\_PROMISE" referencing an origin for which it has not yet received the server's certificate, the client MUST verify the server's possession of an appropriate certificate by sending a "CERTIFICATE\_NEEDED" frame on the pushed stream to inform the server that progress is blocked until the request is satisfied. The client MUST NOT use the pushed resource until an appropriate certificate has been received and validated.



### **3. Certificates Frames for HTTP/2**

The "CERTIFICATE\_REQUEST" and "CERTIFICATE\_NEEDED" frames are correlated by their "Request-ID" field. Subsequent "CERTIFICATE\_NEEDED" frames with the same "Request-ID" value MAY be sent on other streams where the sender is expecting a certificate with the same parameters.

The "CERTIFICATE", and "USE\_CERTIFICATE" frames are correlated by their "Cert-ID" field. Subsequent "USE\_CERTIFICATE" frames with the same "Cert-ID" MAY be sent in response to other "CERTIFICATE\_NEEDED" frames and refer to the same certificate.

"Request-ID" and "Cert-ID" are sender-local, and the use of the same value by the other peer does not imply any correlation between their frames.

#### **3.1. The CERTIFICATE\_NEEDED frame**

The "CERTIFICATE\_NEEDED" frame (0xFRAME-TBD1) is sent to indicate that the HTTP request on the current stream is blocked pending certificate authentication. The frame includes a request identifier which can be used to correlate the stream with a previous "CERTIFICATE\_REQUEST" frame sent on stream zero. The "CERTIFICATE\_REQUEST" describes the certificate the sender requires to make progress on the stream in question.

The "CERTIFICATE\_NEEDED" frame contains 1 octet, which is the authentication request identifier, "Request-ID". A peer that receives a "CERTIFICATE\_NEEDED" of any other length MUST treat this as a stream error of type "PROTOCOL\_ERROR". Frames with identical request identifiers refer to the same "CERTIFICATE\_REQUEST".

A server MAY send multiple "CERTIFICATE\_NEEDED" frames on the same stream. If a server requires that a client provide multiple certificates before authorizing a single request, each required certificate MUST be indicated with a separate "CERTIFICATE\_NEEDED" frame, each of which MUST have a different request identifier (referencing different "CERTIFICATE\_REQUEST" frames describing each required certificate). To reduce the risk of client confusion, servers SHOULD NOT have multiple outstanding "CERTIFICATE\_NEEDED" frames on the same stream at any given time.

Clients MUST NOT send multiple "CERTIFICATE\_NEEDED" frames on the same stream.

The "CERTIFICATE\_NEEDED" frame MUST NOT be sent to a peer which has not advertised support for HTTP-layer certificate authentication.



The "CERTIFICATE\_NEEDED" frame MUST NOT be sent on stream zero, and MUST NOT be sent on a stream in the "half-closed (local)" state [[RFC7540](#)]. A client that receives a "CERTIFICATE\_NEEDED" frame on a stream which is not in a valid state SHOULD treat this as a stream error of type "PROTOCOL\_ERROR".

### **3.2. The USE\_CERTIFICATE Frame**

The "USE\_CERTIFICATE" frame (0xFRAME-TBD4) is sent in response to a "CERTIFICATE\_NEEDED" frame to indicate which certificate is being used to satisfy the requirement.

A "USE\_CERTIFICATE" frame with no payload refers to the certificate provided at the TLS layer, if any. If no certificate was provided at the TLS layer, the stream should be processed with no authentication, likely returning an authentication-related error at the HTTP level (e.g. 403) for servers or routing the request to a new connection for clients.

Otherwise, the "USE\_CERTIFICATE" frame contains the "Cert-ID" of the certificate the sender wishes to use. This MUST be the ID of a certificate for which proof of possession has been presented in a "CERTIFICATE" frame. Recipients of a "USE\_CERTIFICATE" frame of any other length MUST treat this as a stream error of type "PROTOCOL\_ERROR". Frames with identical certificate identifiers refer to the same certificate chain.

The "USE\_CERTIFICATE" frame MUST NOT be sent on stream zero or a stream on which a "CERTIFICATE\_NEEDED" frame has not been received. Receipt of a "USE\_CERTIFICATE" frame in these circumstances SHOULD be treated as a stream error of type "PROTOCOL\_ERROR". Each "USE\_CERTIFICATE" frame should reference a preceding "CERTIFICATE" frame. Receipt of a "USE\_CERTIFICATE" frame before the necessary frames have been received on stream zero MUST also result in a stream error of type "PROTOCOL\_ERROR".

The referenced certificate chain MUST conform to the requirements expressed in the "CERTIFICATE\_REQUEST" to the best of the sender's ability. Specifically:

- o If the "CERTIFICATE\_REQUEST" contained a non-empty "Certificate-Authorities" element, one of the certificates in the chain SHOULD be signed by one of the listed CAs.
- o If the "CERTIFICATE\_REQUEST" contained a non-empty "Cert-Extensions" element, the end-entity certificate MUST match with regard to the extension OIDs recognized by the sender.



If these requirements are not satisfied, the recipient MAY at its discretion either return an error at the HTTP semantic layer, or respond with a stream error [RFC7540] on any stream where the certificate is used. [Section 4](#) defines certificate-related error codes which might be applicable.

### 3.3. The CERTIFICATE\_REQUEST Frame

TLS 1.3 defines the "CertificateRequest" message, which prompts the client to provide a certificate which conforms to certain properties specified by the server. This draft defines the "CERTIFICATE\_REQUEST" frame (0xFRAME-TBD2), which contains the same contents as a TLS 1.3 "CertificateRequest" message, but can be sent over any TLS version.

The "CERTIFICATE\_REQUEST" frame SHOULD NOT be sent to a peer which has not advertised support for HTTP-layer certificate authentication.

The "CERTIFICATE\_REQUEST" frame MUST be sent on stream zero. A "CERTIFICATE\_REQUEST" frame received on any other stream MUST be rejected with a stream error of type "PROTOCOL\_ERROR".

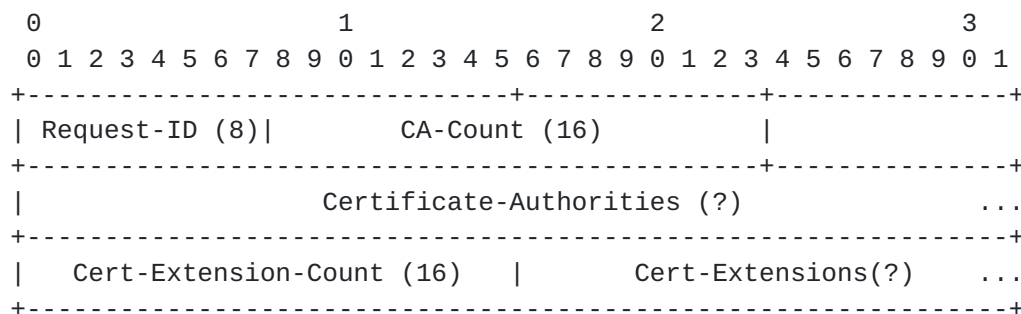


Figure 7: CERTIFICATE\_REQUEST frame payload

The frame contains the following fields:

**Request-ID:** "Request-ID" is an 8-bit opaque identifier used to correlate subsequent certificate-related frames with this request. The identifier MUST be unique in the session for the sender.

**CA-Count and Certificate-Authorities:** "Certificate-Authorities" is a series of distinguished names of acceptable certificate authorities, represented in DER-encoded [X690] format. These distinguished names may specify a desired distinguished name for a root CA or for a subordinate CA; thus, this message can be used to describe known roots as well as a desired authorization space. The number of such structures is given by the 16-bit "CA-Count" field, which MAY be zero. If the "CA-Count" field is zero, then





the recipient MAY send any certificate that meets the rest of the selection criteria in the "CERTIFICATE\_REQUEST", unless there is some external arrangement to the contrary.

**Cert-Extension-Count and Cert-Extensions:** A list of certificate extension OIDs [[RFC5280](#)] with their allowed values, represented in a series of "CertificateExtension" structures (see [[I-D.ietf-tls-tls13](#)] [section 6.3.5](#)). The list of OIDs MUST be used in certificate selection as described in [[I-D.ietf-tls-tls13](#)]. The number of Cert-Extension structures is given by the 16-bit "Cert-Extension-Count" field, which MAY be zero.

Some certificate extension OIDs allow multiple values (e.g. Extended Key Usage). If the sender has included a non-empty Cert-Extensions list, the certificate MUST contain all of the specified extension OIDs that the recipient recognizes. For each extension OID recognized by the recipient, all of the specified values MUST be present in the certificate (but the certificate MAY have other values as well). However, the recipient MUST ignore and skip any unrecognized certificate extension OIDs.

Servers MUST be able to recognize the "subjectAltName" extension ([[RFC2459](#)] [section 4.2.1.7](#)) at a minimum. Clients MUST always specify the desired origin using this extension, though other extensions MAY also be included.

PKIX RFCs define a variety of certificate extension OIDs and their corresponding value types. Depending on the type, matching certificate extension values are not necessarily bitwise-equal. It is expected that implementations will rely on their PKI libraries to perform certificate selection using these certificate extension OIDs.

### **[3.4.](#) The CERTIFICATE Frame**

The "CERTIFICATE" frame (id=0xFRAME-TBD3) provides a exported authenticator message ([[I-D.sullivan-tls-exported-authenticator](#)]) from the TLS layer that provides a chain of certificates, associated extensions and proves possession of the private key corresponding to the end-entity certificate.

The "CERTIFICATE" frame defines one flag:

**AUTOMATIC\_USE (0x01):** Indicates that the certificate can be used automatically on future requests.





Figure 8: CERTIFICATE frame payload

The "CERTIFICATE" frame (0xFRAME-TBD4) contains an "Exported Authenticator" field which corresponds to the value returned from the TLS connection exported authenticator API when provided with a certificate, a valid certificate chain for the connection and associated extensions (OCSP, SCT, etc.), and a connection-unique 8-byte certificate\_request\_context value.

If the "AUTOMATIC\_USE" flag is set, the recipient MAY omit sending "CERTIFICATE\_NEEDED" frames on future streams which would require a similar certificate and use the referenced certificate for authentication without further notice to the holder. This behavior is optional, and receipt of a "CERTIFICATE\_NEEDED" frame does not imply that previously-presented certificates were unacceptable, even if "AUTOMATIC\_USE" was set. Servers MUST set the "AUTOMATIC\_USE" flag when sending a "CERTIFICATE" frame. A server MUST NOT send certificates for origins which it is not prepared to service on the current connection.

Upon receiving a CERTIFICATE frame, the receiver may validate the Exported Authenticator value by using the exported authenticator API. This returns either an error indicating that the message was invalid, or the certificate chain and extensions used to create the message.

It is possible that a large certificate chain might be too large to fit in a single HTTP/2 frame (see [\[RFC7540\] section 4.2](#)). Senders unable to transfer a requested certificate due to the recipient's "SETTINGS\_MAX\_FRAME\_SIZE" value SHOULD terminate affected streams with "CERTIFICATE\_TOO\_LARGE".

The "CERTIFICATE" frame MUST be sent on stream zero. A "CERTIFICATE" frame received on any other stream MUST be rejected with a stream error of type "PROTOCOL\_ERROR".

#### **4. Indicating failures during HTTP-Layer Certificate Authentication**

Because this draft permits certificates to be exchanged at the HTTP framing layer instead of the TLS layer, several certificate-related errors which are defined at the TLS layer might now occur at the HTTP framing layer. In this section, those errors are restated and added to the HTTP/2 error code registry.



BAD\_CERTIFICATE (0xERROR-TBD1): A certificate was corrupt, contained signatures that did not verify correctly, etc.

UNSUPPORTED\_CERTIFICATE (0xERROR-TBD2): A certificate was of an unsupported type or did not contain required extensions

CERTIFICATE\_REVOKED (0xERROR-TBD3): A certificate was revoked by its signer

CERTIFICATE\_EXPIRED (0xERROR-TBD4): A certificate has expired or is not currently valid

CERTIFICATE\_TOO\_LARGE (0xERROR-TBD5): The certificate cannot be transferred due to the recipient's "SETTINGS\_MAX\_FRAME\_SIZE"

CERTIFICATE\_GENERAL (0xERROR-TBD6): Any other certificate-related error

As described in [[RFC7540](#)], implementations MAY choose to treat a stream error as a connection error at any time. Of particular note, a stream error cannot occur on stream 0, which means that implementations cannot send non-session errors in response to "CERTIFICATE\_REQUEST", and "CERTIFICATE" frames. Implementations which do not wish to terminate the connection MAY either send relevant errors on any stream which references the failing certificate in question or process the requests as unauthenticated and provide error information at the HTTP semantic layer.

## **5. Security Considerations**

This mechanism defines an alternate way to obtain server and client certificates other than in the initial TLS handshake. While the signature of exported authenticator values is expected to be equally secure, it is important to recognize that a vulnerability in this code path is at least equal to a vulnerability in the TLS handshake.

### **5.1. Impersonation**

This mechanism could increase the impact of a key compromise. Rather than needing to subvert DNS or IP routing in order to use a compromised certificate, a malicious server now only needs a client to connect to some HTTPS site under its control in order to present the compromised certificate. Clients **MUST** continue to validate that the destination IP address is valid for the origin either by direct DNS resolution or resolution of a validated Alternative Service. (Future work could include a mechanism for a server to offer proofs.)



As noted in the Security Considerations of [\[I-D.sullivan-tls-exported-authenticator\]](#), it is difficult to formally prove that an endpoint is jointly authoritative over multiple certificates, rather than individually authoritative on each certificate. As a result, clients MUST NOT assume that because one origin was previously colocated with another, those origins will be reachable via the same endpoints in the future. Clients MUST NOT consider previous secondary certificates to be validated after TLS session resumption. However, clients MAY query for previously-presented secondary certificates after first repeating the validation of the endpoint's IP address against fresh DNS and Alt-Svc information.

## **5.2. Fingerprinting**

This draft defines a mechanism which could be used to probe servers for origins they support, but opens no new attack versus making repeat TLS connections with different SNI values. Servers SHOULD impose similar denial-of-service mitigations (e.g. request rate limits) to "CERTIFICATE\_REQUEST" frames as to new TLS connections.

While the "CERTIFICATE\_REQUEST" frame permits the sender to enumerate the acceptable Certificate Authorities for the requested certificate, it might not be prudent (either for security or data consumption) to include the full list of trusted Certificate Authorities in every request. Senders, particularly clients, SHOULD send an empty "Certificate-Authorities" element unless they are expecting a certificate to be signed by a particular CA or one of a small set of CAs.

## **5.3. Denial of Service**

Failure to provide a certificate on a stream after receiving "CERTIFICATE\_NEEDED" blocks processing, and SHOULD be subject to standard timeouts used to guard against unresponsive peers.

Validating a multitude of signatures can be computationally expensive, while generating an invalid signature is computationally cheap. Implementations will require checks for attacks from this direction. Signature proofs SHOULD NOT be validated until a stream requires the certificate to make progress. A signature which is not valid based on the asserted public key SHOULD be treated as a session error, to avoid further attacks from the peer, though an implementation MAY instead disable HTTP-layer certificates for the current connection instead.





#### **5.4. Confusion About State**

Implementations need to be aware of the potential for confusion about the state of a connection. The presence or absence of a validated certificate can change during the processing of a request, potentially multiple times, as "USE\_CERTIFICATE" frames are received. A server that uses certificate authentication needs to be prepared to reevaluate the authorization state of a request as the set of certificates changes.

Client implementations need to carefully consider the impact of setting the "AUTOMATIC\_USE" flag. This flag is a performance optimization, permitting the client to avoid a round-trip on each request where the server checks for certificate authentication. However, once this flag has been sent, the client has zero knowledge about whether the server will use the referenced cert for any future request, or even for an existing request which has not yet completed. Clients **MUST NOT** set this flag on any certificate which is not appropriate for currently-in-flight requests, and **MUST NOT** make any future requests on the same connection which they are not willing to have associated with the provided certificate.

### **6. IANA Considerations**

This draft adds entries in three registries.

The HTTP/2 "SETTINGS\_HTTP\_CERT\_AUTH" setting is registered in [Section 6.1](#). Four frame types are registered in [Section 6.2](#). Six error codes are registered in [Section 6.3](#).

#### **6.1. HTTP/2 SETTINGS\_HTTP\_CERT\_AUTH Setting**

The SETTINGS\_HTTP\_CERT\_AUTH setting is registered in the "HTTP/2 Settings" registry established in [[RFC7540](#)].

Name: SETTINGS\_HTTP\_CERT\_AUTH

Code: 0xSETTING-TBD

Initial Value: 0

Specification: This document.

#### **6.2. New HTTP/2 Frames**

Four new frame types are registered in the "HTTP/2 Frame Types" registry established in [[RFC7540](#)]. The entries in the following table are registered by this document.



Frame Type	Code	Specification
CERTIFICATE_NEEDED	0xFRAME-TBD1	<a href="#">Section 3.1</a>
CERTIFICATE_REQUEST	0xFRAME-TBD2	<a href="#">Section 3.3</a>
CERTIFICATE	0xFRAME-TBD3	<a href="#">Section 3.4</a>
USE_CERTIFICATE	0xFRAME-TBD4	<a href="#">Section 3.2</a>

### 6.3. New HTTP/2 Error Codes

Five new error codes are registered in the "HTTP/2 Error Code" registry established in [[RFC7540](#)]. The entries in the following table are registered by this document.

Name	Code	Specification
BAD_CERTIFICATE	0xERROR-TBD1	<a href="#">Section 4</a>
UNSUPPORTED_CERTIFICATE	0xERROR-TBD2	<a href="#">Section 4</a>
CERTIFICATE_REVOKED	0xERROR-TBD3	<a href="#">Section 4</a>
CERTIFICATE_EXPIRED	0xERROR-TBD4	<a href="#">Section 4</a>
CERTIFICATE_GENERAL	0xERROR-TBD5	<a href="#">Section 4</a>

## 7. Acknowledgements

Eric Rescorla pointed out several failings in an earlier revision. Andrei Popov contributed to the TLS considerations.

## 8. References

### 8.1. Normative References

[I-D.ietf-tls-tls13]

Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", [draft-ietf-tls-tls13-19](#) (work in progress), March 2017.



- [I-D.sullivan-tls-exported-authenticator]  
Sullivan, N., "Exported Authenticators in TLS", [draft-sullivan-tls-exported-authenticator-01](#) (work in progress), March 2017.
- [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>>.
- [RFC2459] Housley, R., Ford, W., Polk, W., and D. Solo, "Internet X.509 Public Key Infrastructure Certificate and CRL Profile", [RFC 2459](#), DOI 10.17487/RFC2459, January 1999, <<http://www.rfc-editor.org/info/rfc2459>>.
- [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>>.
- [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, <<http://www.rfc-editor.org/info/rfc5280>>.
- [RFC5705] Rescorla, E., "Keying Material Exporters for Transport Layer Security (TLS)", [RFC 5705](#), DOI 10.17487/RFC5705, March 2010, <<http://www.rfc-editor.org/info/rfc5705>>.
- [RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", [RFC 7230](#), DOI 10.17487/RFC7230, June 2014, <<http://www.rfc-editor.org/info/rfc7230>>.
- [RFC7540] Belshe, M., Peon, R., and M. Thomson, Ed., "Hypertext Transfer Protocol Version 2 (HTTP/2)", [RFC 7540](#), DOI 10.17487/RFC7540, May 2015, <<http://www.rfc-editor.org/info/rfc7540>>.
- [RFC7627] Bhargavan, K., Ed., Delignat-Lavaud, A., Pironti, A., Langley, A., and M. Ray, "Transport Layer Security (TLS) Session Hash and Extended Master Secret Extension", [RFC 7627](#), DOI 10.17487/RFC7627, September 2015, <<http://www.rfc-editor.org/info/rfc7627>>.



- [X690] ITU-T, "Information technology - ASN.1 encoding Rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)", ISO ISO/IEC 8825-1:2002, 2002, <<http://www.itu.int/ITU-T/studygroups/com17/languages/X.690-0207.pdf>>.

## 8.2. Informative References

- [FIPS-186-4] National Institute of Standards and Technology, "Digital Signature Standard (DSS)", FIPS 186-4, July 2013, <<http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf>>.
- [I-D.josefsson-eddsa-ed25519] Josefsson, S. and N. Moller, "EdDSA and Ed25519", [draft-josefsson-eddsa-ed25519-03](#) (work in progress), May 2015.
- [PKCS.1.1991] RSA Laboratories, "RSA Encryption Standard, Version 1.1", PKCS 1, June 1991.
- [RFC2560] Myers, M., Ankney, R., Malpani, A., Galperin, S., and C. Adams, "X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP", [RFC 2560](#), DOI 10.17487/RFC2560, June 1999, <<http://www.rfc-editor.org/info/rfc2560>>.
- [RFC6962] Laurie, B., Langley, A., and E. Kasper, "Certificate Transparency", [RFC 6962](#), DOI 10.17487/RFC6962, June 2013, <<http://www.rfc-editor.org/info/rfc6962>>.
- [RFC7838] Nottingham, M., McManus, P., and J. Reschke, "HTTP Alternative Services", [RFC 7838](#), DOI 10.17487/RFC7838, April 2016, <<http://www.rfc-editor.org/info/rfc7838>>.

## Authors' Addresses

Mike Bishop  
Microsoft

Email: [michael.bishop@microsoft.com](mailto:michael.bishop@microsoft.com)





Nick Sullivan  
Cloudflare

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

Martin Thomson  
Mozilla

Email: [martin.thomson@gmail.com](mailto:martin.thomson@gmail.com)