HTTP
Internet-Draft
Intended status: Standards Track
Expires: November 18, 2016

M. Bishop
Microsoft
M. Thomson
Mozilla

May 17, 2016

Secondary Certificate Authentication in HTTP/2
draft-bishop-httpbis-http2-additional-certs-01

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 a how such certificates can be provided at the HTTP layer to support both scenarios.

Status of This Memo

This Internet-Draft will expire on November 18, 2016.

Copyright Notice

Table of Contents

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.

[I-D.ietf-httpbis-alt-svc] 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 ---->
   <--------------------------- (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 use 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 proposes bringing the TLS 1.3 CertificateRequest, Certificate, and CertificateVerify messages into HTTP/2 frames, enabling certificate-based authentication of both clients and servers independent of TLS version. This mechanism can be implemented at the HTTP layer without requiring new TLS stack behavior and without breaking the existing interface between HTTP and applications above it.

This could be done in a naive manner by replicating the messages as HTTP/2 frames on each stream. However, this would create needless redundancy between streams and require frequent expensive signing operations. Instead, this draft lifts the bulky portions of each message into frames on stream zero and permits the on-stream frames to incorporate them by reference as needed.

Certificate chains, with proof-of-possession of the corresponding private key, 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 is sent as a series of "CERTIFICATE" frames (see Section 3.4) on stream zero. Proof of possession of the corresponding private key is sent as a "CERTIFICATE_PROOF" frame (see Section 3.5) 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_REQUIRED" 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, it uses the setting to communicate its acceptable hash and signature algorithm.

The setting value is a pair of bitmaps. In the lower half, each set bit reflects an acceptable signing algorithm for provided certificates. Each bit MUST NOT be set if a proof signed in this way would be unacceptable to the sender.

Bit 1 (0x00 00 00 01): ECDSA P-256 with SHA-256

Bit 2 (0x00 00 00 02): ECDSA P-384 with SHA-384

Bit 3 (0x00 00 00 04): Ed25519

Bit 4 (0x00 00 00 08): Ed448

Bit 5 (0x00 00 00 10): RSA-PSS with SHA-256 and MGF1 (minimum of 2048 bits)

Bits 6-16: Reserved for future use

If no compatible signature algorithms have been proffered in SETTINGS by a peer, the frames defined in this specification MUST NOT be sent to them, with the exception of empty "USE_CERTIFICATE" frames.

In the upper half, each set bit reflects an acceptable form of supporting data to include with the certificate.

Bit 17 (0x00 01 00 00): Always set. Indicates the ability to interpret requests for certificates.

Bit 18 (0x00 02 00 00): Indicates support for OCSP [RFC2560] supporting data.

Bit 19 (0x00 04 00 00): Indicates support for Signed Certificate Timestamp [RFC6962] supporting data.

Bit 20-32: Reserved for future use

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 --
   <-- (stream 0) CERTIFICATE_PROOF (AU flag) --
   ...
   -- (stream N) GET /from-new-origin --------->
   <----------------------- (stream N) 200 OK --
```

Figure 3: Server-Proffered Certificate

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

Figure 4: Client-Proffered 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_REQUIRED" frame.

Because certificates can be large and each "CERTIFICATE_PROOF" requires a signing operation, the server MAY instead send an "ORIGIN" frame including origins which are not in its TLS certificate. This represents an explicit claim by the server to possess the appropriate certificate - a claim the client MUST verify using the procedures in Section 2.3 before relying on the server's authority for the claimed origin.

2.3. Requiring certificate authentication

As defined in [RFC7540], when a client finds that a https:// origin (or Alternative Service [I-D.ietf-httpbis-alt-svc]) 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_REQUIRED" 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_REQUIRED" 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, 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 and SHOULD send a RST_STREAM on the stream used for the request.

```
Client                                       Server
   <----------------------- (stream 0) ORIGIN --
   -- (stream 0) CERTIFICATE_REQUEST ---------->
   ...
   -- (stream N) CERTIFICATE_REQUIRED --------->
   <--<--<------------ (stream 0) CERTIFICATE --
   <------------ (stream 0) CERTIFICATE_PROOF --
   <-------------- (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_REQUIRED" 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_REQUIRED --
   -- (stream 0) CERTIFICATE ------------>-->-->
   -- (stream 0) CERTIFICATE_PROOF ------------>
   -- (stream N) USE_CERTIFICATE -------------->
   <----------------------- (stream N) 200 OK --
```

Figure 6: Reactive Certificate Authentication

A server MAY push resources from an origin for which it is authoritative but for which the client has not yet received the certificate. In this case, the client MUST verify the server's possession of an appropriate certificate by sending a "CERTIFICATE_REQUIRED" 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_REQUIRED" frames are correlated by their "Request-ID" field. Subsequent "CERTIFICATE_REQUIRED" 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", "CERTIFICATE_PROOF", 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_REQUIRED" 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_REQUIRED frame

The "CERTIFICATE_REQUIRED" frame (0xFRAME-TBD2) 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_REQUIRED" frame contains 1 octet, which is the authentication request identifier, "Request-ID". A peer that receives a "CERTIFICATE_REQUIRED" 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_REQUIRED" 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_REQUIRED" 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_REQUIRED" frames on the same stream at any given time.

Clients MUST NOT send multiple "CERTIFICATE_REQUIRED" frames on the same stream.

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

The "CERTIFICATE_REQUIRED" frame MUST NOT be sent on stream zero, and MUST NOT be sent on a stream in the "half-open (remote)" state. A client that receives a "CERTIFICATE_REQUIRED" 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-TBD5) is sent in response to a "CERTIFICATE_REQUIRED" 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 previously presented in one or more "CERTIFICATE" frames, and for which proof of possession has been presented in a "CERTIFICATE_PROOF" 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_REQUIRED" frame has not been received. Receipt of a "USE_CERTIFICATE" frame in these circmustances SHOULD be treated as 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 first certificate MUST match with regard to the extension OIDs recognized by the sender.

- o Each certificate that is not self-signed MUST be signed using a hash/signature algorithm listed in the "Algorithms" element. [[TODO: No longer exists; does SETTINGS give enough info?]]

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-TBD1), 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".

```
  0                   1                   2                   3
  0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
 +-------------------------------+---------------+---------------+
 | Request-ID (8)|        CA-Count (16)          |
 +-----------------------------------------------+---------------+
 |                   Certificate-Authorities (?)               ...
 +---------------------------------------------------------------+
 |   Cert-Extension-Count (16)   |       Cert-Extensions(?)    ...
 +---------------------------------------------------------------+
```

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

A certificate chain is transferred as a series of "CERTIFICATE" frames (0xFRAME-TBD3) with the same Cert-ID, each containing a single certificate in the chain. The end certificate of the chain can be used as authentication for previous or subsequent requests.

The "CERTIFICATE" frame defines no flags.

While unlikely, it is possible that an exceptionally large certificate 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".

```
  0                   1                   2                   3
  0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
 +-------------------------------+-------------------------------+
 |  Cert-ID (8)  | SData-Count(8)|        SData-Records (*)    ...
 +---------------------------------------------------------------+
 |                          Certificate (*)                    ...
 +---------------------------------------------------------------+
```

Figure 8: CERTIFICATE frame payload

The fields defined by the "CERTIFICATE" frame are:

Cert-ID: The sender-assigned ID of the certificate chain.

SData-Count and SData-Records: An array of Supplemental-Data objects (see Section 3.4.1), with the number given by SData-Count, which MAY be zero. Each Supplemental-Data object contains information about the certificate.

Certificate: An X.509v3 [RFC5280] certificate in the sender's certificate chain.

The first or only "CERTIFICATE" frame with a given Cert-ID MUST contain the sender's certificate. Each subsequent certificate SHOULD directly certify the certificate immediately preceding it. A certificate which specifies a trust anchor MAY be omitted, provided that the recipient is known to already possess the relevant certificate. (For example, because it was included in a "CERTIFICATE_REQUEST"'s Certificate-Authorities list.)

3.4.1. Supplemental-Data

Supplemental data helps a client to validate a certificate, but is not essential to doing so. Peers SHOULD NOT include supplemental data which the recipient is known not to support, but MAY offer supplemental data prior to learning which types the recipient supports.

Each supplemental data object has the following format:

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-------------------------------+-------------------------------+
|     Type(8)   |           Length (16)         |    Data (*) ...
+---------------------------------------------------------------+
```

Figure 9: Supplemental-Data element

The Type field indicates which type of supplemental data is being offered:

OSCP (0x0): Data contains an OCSP [RFC2560] record supporting this certificate.

SCT (0x1): Data contains a Signed Certificate Timestamp [RFC6962] supporting this certificate.

Other values (0x3-0xF): Reserved for future use.

3.5. The CERTIFICATE_PROOF Frame

The "CERTIFICATE_PROOF" frame proves possession of the private key corresponding to an end certificate previously shown in a "CERTIFICATE" frame.

The "CERTIFICATE_PROOF" frame defines one flag:

AUTOMATIC_USE (0x01): Indicates that the certificate can be used automatically on future requests.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-------------------------------+-------------------------------+
|  Cert-ID (8)  |         Algorithm (16)        | Signature(*)...
+---------------------------------------------------------------+
```

Figure 10: CERTIFICATE_PROOF frame payload

The "CERTIFICATE_PROOF" frame (0xFRAME-TBD4) contains an "Algorithm" field (a "SignatureAndHashAlgorithm", from [I-D.ietf-tls-tls13] section 6.3.2.1), describing the hash/signature algorithm pair being used. [[TODO: Sixteen bits because it is in TLS 1.3; if we're using a bitmask to express allowed values, we're down to ~5 bits needed to contain all permitted algorithms. Shrink?]]

The signature is performed as described in [I-D.ietf-tls-tls13], with the following values being used:

o The context string for the signature is "HTTP/2 CERTIFICATE_PROOF"

o The "specified content" is an [RFC5705] exported value, with the following parameters:

  * Disambiguating label string: "EXPORTER HTTP/2 CERTIFICATE_PROOF"

  * Length: 64 bytes

Because the exported value can be independently calculated by both sides of the TLS connection, the value to be signed is not sent on the wire at any time. The same signed value is used for all "CERTIFICATE_PROOF" frames in a single HTTP/2 connection.

A "CERTIFICATE_PROOF" frame MUST be sent only after all "CERTIFICATE" frames with the same Cert-ID have been sent, and MUST correspond to the first certificate presented in the first "CERTIFICATE" frame with that Cert-ID. Receipt of multiple "CERTIFICATE_PROOF" frames for the same Cert-ID, receipt of a "CERTIFICATE_PROOF" frame without a corresponding "CERTIFICATE" frame, or receipt of a "CERTIFICATE" frame after a corresponding "CERTIFICATE_PROOF" MUST be treated as a session error of type "PROTOCOL_ERROR".

If the "AUTOMATIC_USE" flag is set, the recipient MAY omit sending "CERTIFICATE_REQUIRED" 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_REQUIRED" 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_PROOF" frame. A server MUST NOT send certificates for origins which it is not prepared to service on the current connection.

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

BAD_SIGNATURE (0xERROR-TBD5): The digital signature provided did not match the claimed public key

CERTIFICATE_TOO_LARGE (0xERROR-TBD6): The certificate cannot be transferred due to the recipient's "SETTINGS_MAX_FRAME_SIZE"

CERTIFICATE_GENERAL (0xERROR-TBD7): 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", "CERTIFICATE", and "CERTIFICATE_PROOF" 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 the TLS handshake. While the signature of exporter 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.

This could also 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. Clients SHOULD continue to validate that destination IP addresses are 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.)

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, are advised to send an empty "Certificate-Authorities" element unless they are expecting a certificate to be signed by a particular CA or small set of CAs.

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

In order to protect the privacy of the connection against triple-handshake attacks, this feature of HTTP/2 MUST be used only over TLS 1.3 or greater, or over TLS 1.2 in combination with the Extended Master Secret extension defined in [RFC7627].

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.

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.

Finally, validating a multitude of signatures can be computationally expensive, while generating an invalid signature is computationally cheap. Implementations will require checks against 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.

6. IANA Considerations

This draft establishes two new registries, and adds entries in three others.

Acceptable signature methods are registered in Section 6.1. Acceptable forms of supplemental data are registered in Section 6.2.

The HTTP/2 "SETTINGS_HTTP_CERT_AUTH" setting is registered in Section 6.3. Five frame types are registered in Section 6.4. Six error codes are registered in Section 6.5.

6.1. Signature Methods

This document establishes a registry for signature methods acceptable for use with this extension. The "HTTP-Layer Certificate Signature Method" registry manages a space of sixteen values. The "HTTP-Layer Certificate Signature Method" operates under either the "RFC Required" or "IESG Approval" policy.

New entries in this registry require the following information:

Signature Method: A name or label for the signature method

Bit assignment: A single-bit value from 0x0000 to 0x8000

Specification: A document which describes how the signature may be performed

The entries in the following table are registered by this document.

Signature Method	Bit	Specification
ECDSA P-256 with SHA-256	1 (0x0001)	[FIPS-186-4]
ECDSA P-384 with SHA-384	2 (0x0002)	[FIPS-186-4]
Ed25519	3 (0x0004)	[I-D.josefsson-eddsa-ed25519]
Ed448	4 (0x0008)	[I-D.josefsson-eddsa-ed25519]
RSA-PSS with SHA-256 and MGF1	5 (0x0010)	[PKCS.1]

Figure 11

6.2. Supplemental Data

This document establishes a registry for supplemental data types acceptable for use with this extension. The "HTTP-Layer Certificate Supplemental Data" registry manages a space of sixteen values. The "HTTP-Layer Certificate Supplemental Data" operates under either the "RFC Required" or "IESG Approval" policy.

New entries in this registry require the following information:

Data Type: A name or label for the supplemental data type

Bit assignment: A single-bit value from 0x0000 to 0x8000

Value assignment: A value in the range 0x00 to 0xFF; one type MAY reserve multiple values

Specification: A document which describes how the supplemental data may be interpreted

The entries in the following table are registered by this document.

Data Type	Bit	Value	Specification
Reserved	1 (0x0001)	N/A	{{setting}}
OCSP	2 (0x0002)	0x00	[RFC2560]
SCT	3 (0x0004)	0x01	[RFC6962]

Figure 12

6.3. 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.

6.4. New HTTP/2 Frames

Four new frame types are registered in the "HTTP/2 Frame Types" registry established in [RFC7540].

6.4.1. CERTIFICATE_REQUIRED

Frame Type: CERTIFICATE_REQUIRED

Code: 0xFRAME-TBD1

Specification: This document.

6.4.2. CERTIFICATE_REQUEST

Frame Type: CERTIFICATE_REQUEST

Code: 0xFRAME-TBD2

Specification: This document.

6.4.3. CERTIFICATE

Frame Type: CERTIFICATE

Code: 0xFRAME-TBD3

Specification: This document.

6.4.4. CERTIFICATE_PROOF

Frame Type: CERTIFICATE_PROOF

Code: 0xFRAME-TBD4

Specification: This document.

6.4.5. USE_CERTIFICATE

Frame Type: USE_CERTIFICATE

Code: 0xFRAME-TBD5

Specification: This document.

6.5. New HTTP/2 Error Codes

Five new error codes are registered in the "HTTP/2 Error Code" registry established in [RFC7540].

6.5.1. BAD_CERTIFICATE

Name: BAD_CERTIFICATE

Code: 0xERROR-TBD1

Specification: This document.

6.5.2. UNSUPPORTED_CERTIFICATE

Name: UNSUPPORTED_CERTIFICATE

Code: 0xERROR-TBD2

Specification: This document.

6.5.3. CERTIFICATE_REVOKED

Name: CERTIFICATE_REVOKED

Code: 0xERROR-TBD3

Specification: This document.

6.5.4. CERTIFICATE_EXPIRED

Name: CERTIFICATE_EXPIRED

Code: 0xERROR-TBD4

Specification: This document.

6.5.5. BAD_SIGNATURE

Name: BAD_SIGNATURE

Code: 0xERROR-TBD5

Specification: This document.

6.5.6. CERTIFICATE_GENERAL

Name: CERTIFICATE_GENERAL

Code: 0xERROR-TBD6

Specification: This document.

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-tls13-12 (work in progress), March 2016.

[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.ietf-httpbis-alt-svc] Nottingham, M., McManus, P., and J. Reschke, "HTTP Alternative Services", draft-ietf-httpbis-alt-svc-14 (work in progress), March 2016.

[I-D.josefsson-eddsa-ed25519] Josefsson, S. and N. Moller, "EdDSA and Ed25519", draft-josefsson-eddsa-ed25519-03 (work in progress), May 2015.

[I-D.nottingham-httpbis-origin-frame] Nottingham, M. and E. Nygren, "The ORIGIN HTTP/2 Frame", draft-nottingham-httpbis-origin-frame-01 (work in progress), January 2016.

[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>.

Authors' Addresses

Mike Bishop
Microsoft

Email: michael.bishop@microsoft.com

Martin Thomson
Mozilla

Email: martin.thomson@gmail.com