HTTP
Internet-Draft
Intended status: Standards Track
Expires: 8 January 2026

Y. Rosomakho
Zscaler
J. Hoyland
Cloudflare

7 July 2025

Secondary Certificate Authentication of HTTP Clients
draft-rosomakho-httpbis-secondary-client-certs-00

Abstract

This document defines a mechanism for HTTP/2 and HTTP/3 clients to provide additional certificate-based credentials after the TLS handshake has completed, using TLS Exported Authenticators. Unlike traditional client authentication during the TLS handshake, this mechanism allows clients to present multiple certificates over the lifetime of a session.

About This Document

This note is to be removed before publishing as an RFC.

The latest revision of this draft can be found at https://yaroslavros.github.io/httpbis-secondary-client-certs/draft-rosomakho-httpbis-secondary-client-certs.html. Status information for this document may be found at https://datatracker.ietf.org/doc/draft-rosomakho-httpbis-secondary-client-certs/.

Discussion of this document takes place on the HTTP mailing list (mailto:ietf-http-wg@w3.org), which is archived at https://lists.w3.org/Archives/Public/ietf-http-wg/.

Source for this draft and an issue tracker can be found at https://github.com/yaroslavros/httpbis-secondary-client-certs.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

This Internet-Draft will expire on 8 January 2026.

Copyright Notice

Table of Contents

1. Introduction

[TLS] provides an option for client authentication during the initial handshake, but this approach has several limitations. It requires early commitment to a single identity and can expose identifying information prematurely.

This document defines a protocol extension that allows [HTTP/2] and [HTTP/3] clients to provide certificate-based credentials after the TLS handshake, using TLS Exported Authenticators [EXPORTED-AUTH]. This enables clients to authenticate in a flexible and dynamic manner, for example when accessing specific resources, elevating privileges, or switching identities.

An important use case for this mechanism is the ability for a client to provide multiple identities or contextual information distributed across several certificates, such as separate device and user credentials.

This mechanism builds on and complements the secondary certificate support for HTTP servers defined in [SECONDARY-SERVER].

1.1. Authentication Initiation

Client authentication is initiated by the server once both peers have negotiated support for this mechanism.

The client signals support and the maximum number of certificate-based credentials it is expecting to provide during the lifetime of the connection using HTTP/2 or HTTP/3 SETTINGS parameters. The server may request client credentials by sending an AUTHENTICATOR_REQUESTS frame, and the client responds with a sequence of CERTIFICATE frames, each containing an Exported Authenticator.

The server may initiate additional authentication exchanges later in the connection, provided the total number of requested credentials does not exceed the client's declared limit.

1.2. Certificate Validation and Authorization

Client authentication using this mechanism is initiated by the server in response to the client's advertised support.

Servers are expected to evaluate the received credentials according to their policy and decide whether to:

* Accept the certificate and associate it with the client session or with a specific set of future HTTP requests;

* Ignore the certificate and proceed without adjusting the client's authorization state;

* Terminate the connection if the certificate is unacceptable or appears malicious.

This mechanism does not define any explicit protocol-level signaling for certificate rejection. Applications may implement their own logic to determine how authentication success or failure affects access to resources, but such decisions are made outside the scope of this protocol extension.

2. Conventions and Definitions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

3. Negotiating Support for HTTP-layer Client Certificate Authentication

Support for HTTP-layer client certificate authentication is negotiated via a SETTINGS parameter. An endpoint MUST NOT send frames related to client authentication (AUTHENTICATOR_REQUESTS, or client-initiated CERTIFICATE) unless the peer has explicitly advertised support via the SETTINGS parameter defined in Section 3.1.

This restriction does not apply to server-initiated CERTIFICATE frames, which are governed by [SECONDARY-SERVER].

3.1. SETTINGS_HTTP_CLIENT_CERT_AUTH

A new SETTINGS parameter is defined for both HTTP/2 and HTTP/3 to indicate support for HTTP-layer client certificate authentication.

Clients use this parameter to advertise the maximum number of certificate-based credentials they are expecting to provide during the lifetime of the connection. This value MUST be a non-zero integer. A value of zero indicates that the client does not support client certificate authentication.

Servers that support this mechanism MUST respond with a value of 1 to confirm support. The server's value does not affect the number of authenticator requests it can send; it merely confirms participation in the protocol.

Each endpoint must advertise its own support. A client MUST NOT send CERTIFICATE frames unless both parties have advertised support for this mechanism. Similarly, a server MUST NOT send an AUTHENTICATOR_REQUESTS frame unless both parties have advertised support.

Endpoints MUST NOT send a SETTINGS_HTTP_CLIENT_CERT_AUTH parameter with a value of 0 after previously sending a value greater than 0. Once support for this extension has been advertised, it is considered enabled for the lifetime of the connection.

3.1.1. HTTP/2 Definition

In HTTP/2, this parameter is defined as an entry in the SETTINGS frame as specified in Section 6.5.2 of [HTTP/2].

* Identifier: SETTINGS_HTTP_CLIENT_CERT_AUTH (0xTBD)

* Value set by clients: A positive integer indicating the number of credentials they are willing to provide.

* Value set by servers: 1 to indicate support.

3.1.2. HTTP/3 Definition

In HTTP/3, this parameter is defined as a SETTINGS parameter in accordance with Section 7.2.4.1 of [HTTP/3].

* Identifier: SETTINGS_HTTP_CLIENT_CERT_AUTH (0xTBD)

* Value set by clients: A positive integer indicating the number of credentials they are willing to provide.

* Value set by servers: 1 to indicate support.

4. Authentication Procedure

Once both endpoints have negotiated support for HTTP-layer client certificate authentication via SETTINGS_HTTP_CLIENT_CERT_AUTH as described in Section 3.1), the server may initiate client authentication at any time by sending an AUTHENTICATOR_REQUESTS frame.

The AUTHENTICATOR_REQUESTS frame contains a list of authenticator requests, as defined in Section 4 of [EXPORTED-AUTH]. Each request presents a cryptographic challenge for the client to use in the construction of a corresponding Exported Authenticator.

The client MAY use provided authenticator requests to send CERTIFICATE frames. Each CERTIFICATE frame carries an Exported Authenticator that includes a certificate chain and proof of possession, as specified in Section 5 of [EXPORTED-AUTH]. A client MAY send an empty authenticator as defined in Section 6 of [EXPORTED-AUTH] in cases where it declines to authenticate for a given request.

This section defines the structure of the AUTHENTICATOR_REQUESTS frame and outlines how it and the CERTIFICATE frames, as defined in [SECONDARY-SERVER], are exchanged, ordered, and validated during the authentication process.

4.1. AUTHENTICATOR_REQUESTS Frame

The AUTHENTICATOR_REQUESTS frame is sent by the server to deliver authenticator requests to the client. It may be sent at any point after the client has advertised support via the SETTINGS_HTTP_CLIENT_CERT_AUTH parameter.

The server MAY send multiple AUTHENTICATOR_REQUESTS frames over the lifetime of the connection. However, the total number of outstanding authenticator requests (those for which the client has not yet responded with a CERTIFICATE frame) MUST NOT exceed the maximum specified by the client in its SETTINGS_HTTP_CLIENT_CERT_AUTH parameter. If this limit is exceeded, the client MUST treat it as a connection error.

The server MAY send additional AUTHENTICATOR_REQUESTS frames after receiving one or more CERTIFICATE frames from the client, in order to replenish the client’s pool of active challenges. This allows the server to control the pace of client authentication while adhering to the negotiated limits.

The list of authenticator requests in AUTHENTICATOR_REQUESTS frame MUST NOT be empty.

Each authenticator request is used by the client to construct an Exported Authenticator as defined in Section 5 of [EXPORTED-AUTH]. This frame is a control frame and MUST be sent on stream 0 in HTTP/2 or the control stream in HTTP/3.

4.1.1. HTTP/2 Definition

```
AUTHENTICATOR_REQUESTS Frame {
  Length (24),
  Type (8) = 0xTBD,
  Unused Flags (8),

  Reserved (1),
  Stream Identifier (31) = 0,

  Authenticator Request (..) ...,
}
```

Figure 1: HTTP/2 AUTHENTICATOR_REQUESTS Frame

If this frame is received by a server on any stream, or by a client on a stream other than stream 0, the recipient MUST treat it as a connection error of type PROTOCOL_ERROR according to Section 5.4.1 of [HTTP/2].

If the frame payload is malformed or contains malformed authenticator requests (e.g., incomplete length-prefixes or invalid encoding), the recipient MUST treat it as a connection error of type PROTOCOL_ERROR.

4.1.2. HTTP/3 Definition

```
AUTHENTICATOR_REQUESTS Frame {
  Type (i) = 0xTBD,
  Length (i),

  Authenticator Request (..) ...,
}
```

Figure 2: HTTP/3 AUTHENTICATOR_REQUESTS Frame

If this frame is received by a server on any stream, or by a client on a stream other than the control stream, the recipient MUST treat it as a connection error of type H3_FRAME_UNEXPECTED according to Section 8 of [HTTP/3].

If the frame payload is malformed or contains malformed authenticator requests, the recipient MUST treat it as a connection error of type H3_MESSAGE_ERROR.

4.1.3. Authenticator Request Format

Each Authenticator Request is an encoded structure consisting of a variable-length integer length and the corresponding binary value of the request:

```
Authenticator Request {
  Length (i),
  ClientCertificateRequest (..),
}
```

Figure 3: Authenticator Request structure

* Length (i): A variable-length integer specifying the length, in bytes, of the ClientCertificateRequest.

* ClientCertificateRequest: A structure as defined in Section 4 of [EXPORTED-AUTH].

The payload of the AUTHENTICATOR_REQUESTS frame consists of a sequence of these Authenticator Request elements. The list MUST NOT be empty. Clients parse the payload by reading the length, then interpreting the following bytes as a ClientCertificateRequest structure.

4.1.4. Request Ordering and Pacing

Clients MUST respond to each authenticator request context with a corresponding CERTIFICATE frame, in the same order as the authenticator requests appeared in the AUTHENTICATOR_REQUESTS frame.

The total number of outstanding authenticator requests MUST NOT exceed the value the client advertised in the SETTINGS_HTTP_CLIENT_CERT_AUTH parameter. If the server exceeds this limit, the client MUST treat it as a connection error.

In HTTP/2, the connection MUST be closed with a PROTOCOL_ERROR according to Section 5.4.1 of [HTTP/2]. In HTTP/3, the connection MUST be closed with H3_FRAME_UNEXPECTED according to Section 8 of [HTTP/3].

4.2. CERTIFICATE Frame

The CERTIFICATE frame is used by the client to respond to each authenticator request previously delivered by the server in an AUTHENTICATOR_REQUESTS frame. Each CERTIFICATE frame carries a single Exported Authenticator structure constructed using the TLS Exported Authenticator mechanism defined in Section 5.2 of [EXPORTED-AUTH].

The format and semantics of the CERTIFICATE frame are defined in [SECONDARY-SERVER], and are reused without modification for client authentication.

Clients MUST send exactly one CERTIFICATE frame for each authenticator request context received. These frames MUST be sent in the same order as the corresponding authenticator requests appeared in the AUTHENTICATOR_REQUESTS frame.

A CERTIFICATE frame MAY carry an empty authenticator, as described in Section 6 of [EXPORTED-AUTH]. This allows the client to decline authentication for a particular request context without terminating the connection.

Clients MUST NOT send CERTIFICATE frames unless they are in response to an outstanding AUTHENTICATOR_REQUESTS frame. If a server receives a CERTIFICATE frame when no authenticator request is outstanding, it MUST treat this as a connection error.

Servers MUST NOT send CERTIFICATE frames unless the secondary certificate authentication mechanism for HTTP servers has been negotiated according to [SECONDARY-SERVER]. If a client receives a server-generated CERTIFICATE frame without such negotiation, it MUST treat this as a connection error of type PROTOCOL_ERROR (HTTP/2) or H3_FRAME_UNEXPECTED (HTTP/3).

4.2.1. Certificate Validation and Authorization

The validation of client-provided certificates and any associated authorization logic are outside the scope of this specification. The server is responsible for interpreting the CERTIFICATE frames it receives and deciding how they impact the session state or access control.

A server MAY associate an accepted certificate with the connection, use it for access control decisions on subsequent requests, or ignore it entirely. The application determines whether authentication is required, optional, or sufficient for specific operations.

If a server receives a certificate that it considers invalid, untrusted, or inappropriate, it MAY respond by limiting the client's access, rejecting individual requests, or closing the connection. This specification does not define protocol-level error signaling for certificate rejection.

5. Security Considerations

This mechanism builds on TLS Exported Authenticators [EXPORTED-AUTH], which provide cryptographic binding to the underlying TLS connection and strong guarantees of authenticity, integrity, and freshness. However, several considerations remain relevant for implementers and deployers.

5.1. Impersonation and Trust

Servers MUST carefully validate client-provided certificates and ensure that they are trusted, appropriate for the context, and bound to the intended identity. Failure to enforce proper certificate validation could allow an attacker to impersonate another client or gain unauthorized access.

Authorization decisions based on certificates are application-specific and must be handled with care. A certificate's presence does not imply entitlement.

5.2. Replay Prevention

Exported Authenticators are constructed using unique per-session cryptographic context and include binding to the TLS connection via the handshake transcript. This provides replay protection across different sessions, provided the ClientCertificateRequest includes a fresh context value. Reuse of identical request contexts across connections SHOULD be avoided.

5.3. Exposure of Client Identity

Certificates often contain long-lived identifiers (such as names, device serial numbers, or email addresses). When a client presents such certificates, it exposes identity information to the server. Clients should avoid presenting identifying certificates unless necessary for the resource or context being accessed.

If client authentication is not required, clients may prefer to respond with an empty Exported Authenticator.

5.4. Binding to the Connection

Exported Authenticators are cryptographically bound to the TLS session in which they are sent. A certificate authenticated using this mechanism applies to the entire HTTP connection and all streams within it. Servers MUST NOT assume that certificate-based identity can vary between streams on the same connection.

Applications MUST ensure that any authenticated identity is scoped to the connection lifetime, and not reused across connections unless explicitly permitted by policy.

5.5. Denial-of-Service Risk

The process of generating authenticator requests, tracking their state, and verifying the resulting authenticators can be computationally expensive for the server. A malicious or misbehaving client could send excessive CERTIFICATE frames to force the server to consume memory, perform cryptographic operations, or maintain authentication state unnecessarily.

Servers SHOULD enforce limits on the number of authenticator requests they are willing to issue. Servers MAY respond with fewer Authenticator Requests than requested by the client or choose not to repelish consumed requests with new ones. Implementations SHOULD bound memory and compute usage per connection to mitigate such attacks.

5.6. Timing Side Channels

Applications should consider the potential for timing side channels in certificate validation and authenticator generation. Variations in processing time may reveal information about internal policies, trust anchors, or user behavior. Constant-time validation and randomized delays are potential mitigations in sensitive deployments.

6. IANA Considerations

This document registers the AUTHENTICATOR_REQUESTS frame type and SETTINGS_HTTP_CLIENT_CERT_AUTH for both [HTTP/2] and [HTTP/3].

6.1. HTTP/2 Frame Types

This specification registers the following entry in the "HTTP/2 Frame Types" registry defined in [HTTP/2]:

AUTHENTICATOR_REQUESTS frame:

* Code: 0xTBD

* Frame Type: AUTHENTICATOR_REQUESTS

* Reference: This document

6.2. HTTP/3 Frame Types

This specification registers the following entry in the "HTTP/3 Frame Types" registry defined in [HTTP/3]:

AUTHENTICATOR_REQUESTS frame:

* Value: 0xTBD

* Frame Type: AUTHENTICATOR_REQUESTS

* Status: provisional (permanent if this document is approved)

* Reference: This document

* Change Controller: Yaroslav Rosomakho (IETF if this document is approved)

* Contact: yrosomakho@zscaler.com (HTTP_WG; HTTP working group; ietf-http-wg@w3.org if this document is approved)

* Notes: None

6.3. HTTP/2 Setting

This specification registers the following entry in the "HTTP/2 Settings" registry defined in [HTTP/2]:

* Code: 0xTBD

* Name: SETTINGS_HTTP_CLIENT_CERT_AUTH

* Initial Value: 0

* Reference: This document

6.4. HTTP/3 Setting

This specification registers the following entry in the "HTTP/3 Settings" registry defined in [HTTP/3]:

* Code: 0xTBD

* Setting Name: SETTINGS_HTTP_CLIENT_CERT_AUTH

* Default: 0

* Status: provisional (permanent if this document is approved)

* Reference: This document

* Change Controller: Yaroslav Rosomakho (IETF if this document is approved)

* Contact: yrosomakho@zscaler.com (HTTP_WG; HTTP working group; ietf-http-wg@w3.org if this document is approved)

* Notes: None

7. Normative References

[EXPORTED-AUTH] Sullivan, N., "Exported Authenticators in TLS", RFC 9261, DOI 10.17487/RFC9261, July 2022, <https://www.rfc-editor.org/rfc/rfc9261>.

[HTTP/2] Thomson, M., Ed. and C. Benfield, Ed., "HTTP/2", RFC 9113, DOI 10.17487/RFC9113, June 2022, <https://www.rfc-editor.org/rfc/rfc9113>.

[HTTP/3] Bishop, M., Ed., "HTTP/3", RFC 9114, DOI 10.17487/RFC9114, June 2022, <https://www.rfc-editor.org/rfc/rfc9114>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/rfc/rfc2119>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/rfc/rfc8174>.

[SECONDARY-SERVER] Gorbaty, E. and M. Bishop, "Secondary Certificate Authentication of HTTP Servers", Work in Progress, Internet-Draft, draft-ietf-httpbis-secondary-server-certs-01, 12 October 2024, <https://datatracker.ietf.org/doc/html/draft-ietf-httpbis-secondary-server-certs-01>.

[TLS] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <https://www.rfc-editor.org/rfc/rfc8446>.

Acknowledgments

TODO acknowledge.

Authors' Addresses

Yaroslav Rosomakho
Zscaler
Email: yrosomakho@zscaler.com

Jonathan Hoyland
Cloudflare
Email: jonathan.hoyland@gmail.com