Transport Layer Security
Internet-Draft
Intended status: Standards Track
Expires: 25 June 2026

Y. Rosomakho
Zscaler
T. Reddy
Nokia

22 December 2025

Certificate Update in TLS 1.3
draft-rosomakho-tls-cert-update-01

Abstract

This document defines a mechanism that enables TLS 1.3 endpoints to update their certificates during the lifetime of a connection using Exported Authenticators. A new extension is introduced to negotiate support for certificate update at handshake time. When negotiated, either endpoint can provide a post-handshake authenticator containing an updated certificate, delivered via a new handshake message. This mechanism allows long-lived TLS connections to remain valid across certificate rotations without requiring session termination.

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/draft-tls-cert-refresh/draft-rosomakho-tls-cert-update.html. Status information for this document may be found at https://datatracker.ietf.org/doc/draft-rosomakho-tls-cert-update/.

Discussion of this document takes place on the Transport Layer Security mailing list (mailto:tls@ietf.org), which is archived at https://mailarchive.ietf.org/arch/browse/tls/. Subscribe at https://www.ietf.org/mailman/listinfo/tls/.

Source for this draft and an issue tracker can be found at https://github.com/yaroslavros/draft-tls-cert-refresh.

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 25 June 2026.

Copyright Notice

Table of Contents

1. Introduction

[TLS] provides strong guarantees of confidentiality, integrity, and authentication, but does not include a general-purpose mechanism for updating certificates of both endpoints after the handshake. While TLS 1.3 permits post-handshake authentication for clients as defined in Section 4.6.2 of [TLS], it provides no standardized way for servers to update their certificates once the handshake is complete. Additionally, TLS 1.3 post-handshake authentication is explicitly prohibited from QUIC according to Section 4.4 of [QUIC-TLS] and HTTP/2 according to Section 2 of [HTTP2-TLS13] because it allows clients to introduce new certificates and change authorization properties of the connection.

This presents a limitation for long-lived connections in environments where certificates may need to be refreshed, whether due to expiration, revocation, or key rotation. Terminating and re-establishing connections solely for the purpose of updating certificates can be disruptive and inefficient.

Exported Authenticators [EXPORTED-AUTH] offer a general-purpose mechanism for proving possession of a certificate after the handshake, using an authenticator request supplied by the peer. However, the specification does not define a mechanism for transmitting authenticator requests or delivering authenticators at the TLS layer.

This document defines a mechanism for certificate updates within an established TLS 1.3 connection. It introduces a new extension, certificate_update_request, that allows each endpoint to optionally include an authenticator request as part of the initial handshake. This request can later be used by the peer to generate an Exported Authenticator containing an update certificate.

To deliver the updated certificate, a new TLS handshake message, CertificateUpdate, is defined. This message carries a complete Exported Authenticator and may be sent by either endpoint after the handshake, as long as an authenticator request was previously provided by the peer.

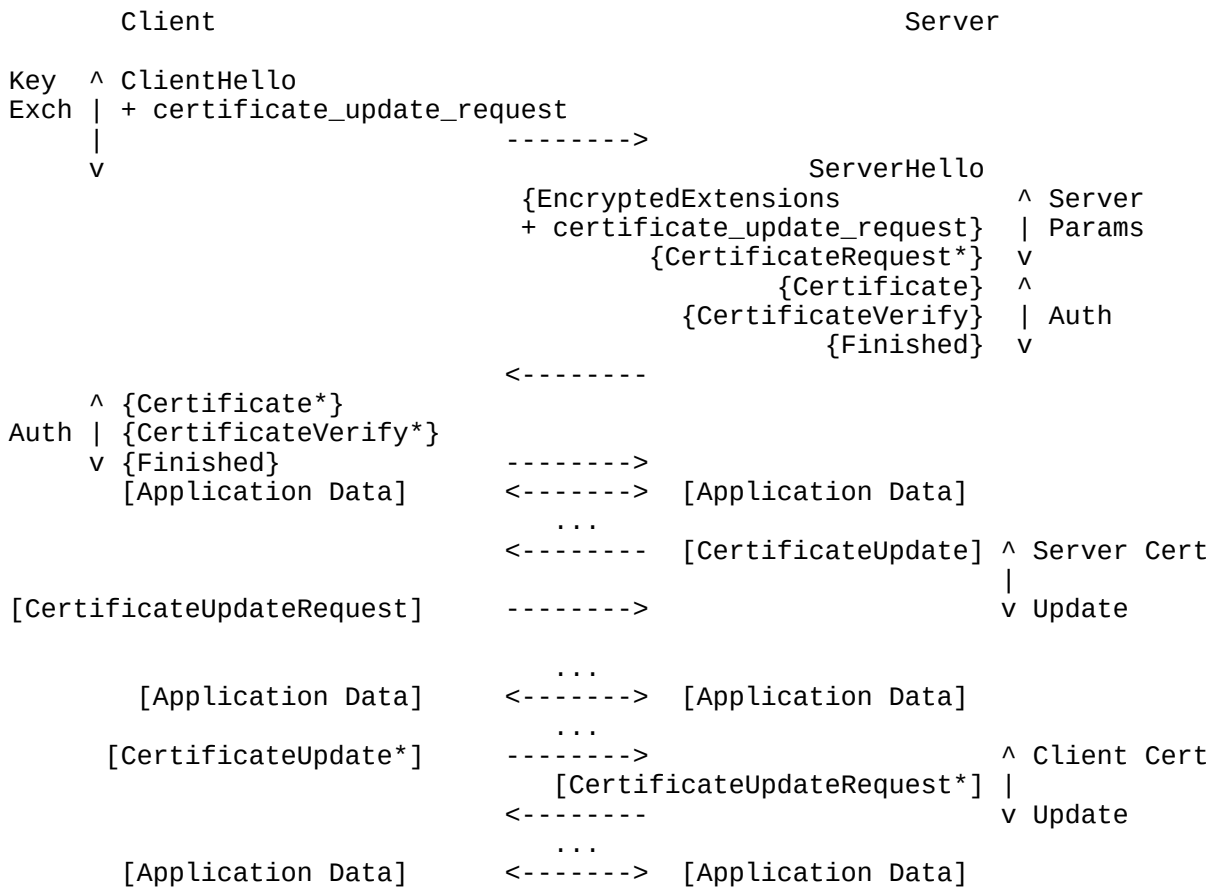
Because authenticator requests are single-use and may not be reused in subsequent authenticator constructions, a second post-handshake message is defined: CertificateUpdateRequest. This message allows an endpoint to provide a new authenticator request for future use after processing a CertificateUpdate.

This approach allows TLS connections to remain valid across certificate updates without requiring session termination. It is compatible with TLS 1.3 and [QUIC], and can be used regardless of the application protocol encapsulated within the connection.

The certificate update mechanism in the document is deliberately constrained to preserve the authentication and authorization context of the connection. The updated certificate must retain the same subject, attributes, and issuing certificate authority as the original, with the only permitted difference being the validity period. This ensures that the peer identity remains unchanged and that application-layer authorization decisions based on the original certificate continue to hold after the update. By limiting the scope of updates in this way, the mechanism provides secure and seamless certificate refresh without altering the security properties of the TLS session.

1.1. Certificate Update flow

Diagram below illustrates the certificate update process:



*Indicates messages/extensions that are only used for client authentication.

Figure 1: Certificate Update Process

TLS peers negotiate support of the Certificate Update mechanism by including certificate_update_request extension in the ClientHello and EncryptedExtensions messages. As explained in Section 3, the extention MAY contain an Authenticator Request can later be used by the peer to provide an updated certificate.

During the lifetime of the TLS session, either peer MAY provide updated certificate by sending a CertificateUpdate message containing an Authenticator, as defined in Section 4. After successfully validating the updated certificate, the receiving peer MAY provide a new Authenticator Request using the CertificateUpdateRequest message defined in Section 5.

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 and Providing Initial Authenticator Request

To enable certificate updates, endpoints must explicitly indicate support during the TLS 1.3 handshake. This is done using the certificate_update_request extension, which may be included in either the ClientHello or EncryptedExtensions message.

The presence of this extension indicates that the sender supports the certificate update mechanism defined in this document. Optionally, the extension MAY carry an authenticator request. If present, this request can be used by the peer to construct an exported authenticator for delivery in a CertificateUpdate handshake message.

The extension MAY also be included with an empty "extension_data" field, which indicates support for the mechanism without offering an authenticator request.

3.1. The certificate_update_request Extension

The certificate_update_request extension is used to negotiate support for post-handshake certificate updates. It MAY appear in the ClientHello and in the EncryptedExtensions message.

The extension MUST NOT appear more than once from the same endpoint during a handshake. If it appears multiple times or in an unexpected message, the receiving peer MUST abort the handshake with an illegal_parameter alert.

* When sent in a ClientHello, the "extension_data" field, MUST be empty or contain a CertificateRequest structure as defined in Section 4 of [EXPORTED-AUTH].

* When sent in EncryptedExtensions, the "extension_data" field, MUST be empty or contain a ClientCertificateRequest structure, also defined in Section 4 of [EXPORTED-AUTH].

If "extension_data" field is not empty, it is encoded as follows:

```
struct {
    select (Handshake.msg_type) {
        case client_hello:         CertificateRequest;
        case encrypted_extensions: ClientCertificateRequest;
    }
} CertificateUpdateRequestExtension;
```

Figure 2: Optional extension_data of certificate_update_request extension

The semantics of the extension are as follows:

* If the extension is omitted, the sender indicates that it does not support receiving or sending certificate updates.

* If the extension is present with a zero-length extension_data, the sender indicates that it does not wish to receive certificate updates, but may provide a CertificateUpdate message later in the session if the peer supplied an authenticator request.

* If the extension is present with a non-empty "extension_data" field, the sender indicates support for certificate updates and provides an authenticator request that the peer may use to generate an exported authenticator in CertificateUpdate message. The extensions list inside the authenticator request MUST be empty.

3.2. Handling Malformed Extension Data

If an endpoint receives a certificate_update_request extension with a non-empty "extension_data" field that cannot be parsed as a valid CertificateRequest (when received in ClientHello) or ClientCertificateRequest (when received in EncryptedExtensions) with empty authenticator request extensions, it MUST abort the handshake with an illegal_parameter alert.

Endpoints MUST NOT attempt to interpret or store a malformed authenticator request. The extension is either valid and usable or invalid and fatal to the handshake.

4. Certificate Update

Once the the handshake has completed and a peer has provided an authenticator request the other endpoint may send a certificate update using the CertificateUpdate handshake message.

The CertificateUpdate message carries a single Exported Authenticator, constructed using the previously received authenticator request. Authenticator requests are single-use and MUST NOT be reused for multiple updates.

The message is structured as follows:

```
struct {
    opaque authenticator<1..2^24-1>;
} CertificateUpdate;
```

Figure 3: CertificateUpdate message

Embedded authenticator is defined in Section 5.2.4 of [EXPORTED-AUTH] as a concantenation of serialized Certificate, CertificateVerify and Finished messages.

4.1. Requirements for Authenticators

The Exported Authenticator carried in a CertificateUpdate message MUST meet the following requirements:

* The subject field of the certificate MUST match exactly with the certificate originally presented by the sender during the handshake.

* All extensions present in the original certificate MUST also be present in the updated certificate with identical values.

* Updated certificate MUST NOT contain any extensions that are not present in the original certificate.

* The public key MAY differ from the original, but the public key algorithm and key length MUST match those of the original certificate.

* The issuer of the updated certificate MUST be the same as the issuer of the original certificate.

* The SignatureScheme used in the CertificateVerify message of the Exported Authenticator MUST be the same as the one used in the sender’ s original handshake authentication.

* The certificate provided in the CertificateUpdate message MUST NOT have been used previously by the sender during the current TLS session.

These constraints ensure that the authenticator represents the same logical identity and cryptographic profile as the original authentication, while ensuring freshness and preventing redundant certificate reuse. A peer that receives a CertificateUpdate containing a certificate or signature that does not meet these requirements MUST terminate the connection with an illegal_parameter alert.

4.2. Sending and Receiving Certificate Updates

A CertificateUpdate message MAY be sent by either endpoint only after the handshake has completed, specifically after the sender has sent and received the Finished message. The message MUST NOT be sent during the handshake or before the authentication block is complete.

A CertificateUpdate message MUST NOT be sent unless a valid authenticator request has been received from the peer and has not yet been used. Each authenticator request is single-use and MUST NOT be reused across multiple updates.

In addition, a client MUST NOT send a CertificateUpdate unless it previously authenticated with a certificate during the handshake or via TLS 1.3 post-handshake authentication Section 4.6.2 of [TLS].

If any of the above conditions are violated, the recipient MUST terminate the connection with an unexpected_message alert.

The message carries a single Exported Authenticator, as defined in Section 5.2.4 of [EXPORTED-AUTH], in the authenticator field. If the authenticator is empty (i.e., it does not contain a Certificate), the peer MUST terminate the connection with an illegal_parameter alert.

Upon receiving a valid CertificateUpdate, the peer MUST validate the Exported Authenticator according to [EXPORTED-AUTH] and Section 4.1 of this document. If validation fails, the connection MUST be aborted with an illegal_parameter alert.

Only one certificate update is permitted per authenticator request. Additional update MUST be provided only in response to a CertificateUpdateRequest message as described in Section 5.

5. Additional Authenticator Requests

Each authenticator request is intended for one-time use, meaning that after an endpoint uses it to generate a CertificateUpdate, it cannot be reused for additional updates. To allow for subsequent certificate updates over the lifetime of a TLS connection, this document defines a new handshake message: CertificateUpdateRequest.

The CertificateUpdateRequest message allows an endpoint to send a fresh authenticator request to the peer after a previous CertificateUpdate has been successfully processed.

The message is structured as follows:

```
struct {
    opaque authenticator_request<1..2^16-1>;
} CertificateUpdateRequest;
```

Figure 4: CertificateUpdateRequest message

The authenticator_request field MUST contain either a CertificateRequest or a ClientCertificateRequest, depending on the role of the sender and the direction of certificate update expected:

* A client sends a CertificateRequest structure (to refresh the server's certificate).

* A server sends a ClientCertificateRequest structure (to refresh the client's certificate).

The structure and encoding of these fields are identical to the formats defined in Section 4 of [EXPORTED-AUTH]. The extensions of the authenticator request MUST be empty.

5.1. Sending and Receiving CertificateUpdateRequest

A CertificateUpdateRequest message MAY be sent at any time after processing a valid CertificateUpdate from the peer. An endpoint MUST NOT send a CertificateUpdateRequest unless it has received and verified a CertificateUpdate using the previous authenticator request. The endpoint MUST wait for the peer to complete a CertificateUpdate using the outstanding request before sending a new one.

Upon receiving a CertificateUpdateRequest, the peer MUST validate that the message contains a well-formed CertificateRequest or ClientCertificateRequest, depending on the sender's role. If the message is malformed, the authenticator request cannot be parsed or the authenticator request contains extensions, the connection MUST be terminated with an illegal_parameter alert.

6. Applicability to QUIC

This specification is applicable to the [QUIC] protocol, which uses TLS 1.3 for connection security as described in [QUIC-TLS].

Endpoints implementing this specification over QUIC MUST encapsulate CertificateUpdate and CertificateUpdateRequest handshake messages within CRYPTO frames, consistent with the general treatment of TLS messages in QUIC.

All other requirements defined in this document, including handshake process, message sequences, validation procedures, and error handling, apply equally to QUIC deployments. A QUIC connection MUST treat protocol violations (such as sending a CertificateUpdate before the handshake completes) as connection errors of type CRYPTO_ERROR, using an appropriate alert code such as illegal_parameter or unexpected_message mapped as defined in Section 20.1 of [QUIC].

The certificate update mechanism defined in this document is compatible with QUIC as it does not introduce changes to the peer’s identity.

7. Applicability to DTLS

This specification is also applicable to DTLS 1.3 [DTLS]. The CertificateUpdate and CertificateUpdateRequest messages are handshake messages and are subject to DTLS built-in support for sequencing, fragmentation, retransmission, and replay detection, as specified in [DTLS]. No additional protection mechanisms are required beyond the normal DTLS handshake processing.

8. Security Considerations

This mechanism relies on the security properties of [TLS] and Exported Authenticators [EXPORTED-AUTH]. It introduces additional opportunities for certificate-based authentication after the initial handshake and requires careful handling to avoid introducing vulnerabilities.

8.1. Identity Continuity

The requirements on updated certificates and signature algorithms ensure that the logical identity authenticated during the handshake remains consistent after a certificate refresh. Endpoints MUST enforce the constraints described in Section 4.1 to prevent identity swapping or privilege escalation.

Updated certificates MUST NOT introduce new names, extensions, or capabilities beyond those present in the original certificate. Failure to enforce these constraints could allow an attacker to impersonate a different entity within the same connection.

8.2. Replay Prevention

Each authenticator request MUST be used exactly once to prevent replay attacks. Exported Authenticators inherently bind to the session and context, but replaying an old CertificateUpdate against a reused authenticator request could allow undetected identity reuse.

Endpoints MUST discard authenticator requests after a successful CertificateUpdate and MUST validate that each certificate update corresponds uniquely to the most recent authenticator request received.

8.3. Application-Layer Implications

Applications that rely on peer certificate properties for access control decisions MAY reevaluate those decisions after a certificate update if needed. However, because the updated certificate is required to maintain the same identity, such re-validation is typically unnecessary for applications that rely only on the peer’s authenticated identity. If the updated certificate does not match the identity validated during the TLS handshake, the TLS stack MUST terminate the connection.

9. IANA Considerations

This document registers a new TLS extension and two new TLS handshake message types, as described below.

9.1. TLS Extension

IANA is requested to add the following entry to the "TLS ExtensionType Values" registry:

* The value is TBD
* The Extension Name is certificate_update_request
* The TLS 1.3 value is CH, EE
* The DTLS-Only value is N
* The Recommended value is Y
* The Reference is this document

9.2. TLS Handshake Message Types

IANA is requested to add the following entries to the "TLS HandshakeType" registry.

CertificateUpdate message:

* The value is TBD1
* The description is certificate_update
* The DTLS-OK is Y
* The Reference is this document
* The Comment section is empty

CertificateUpdateRequest message:

* The value is TBD2
* The description is certificate_update_request
* The DTLS-OK is Y
* The Reference is this document
* The Comment section is empty

10. References

10.1. Normative References

[DTLS] Rescorla, E., Tschofenig, H., and N. Modadugu, "The Datagram Transport Layer Security (DTLS) Protocol Version 1.3", RFC 9147, DOI 10.17487/RFC9147, April 2022, <https://www.rfc-editor.org/rfc/rfc9147>.

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

[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>.

[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>.

10.2. Informative References

[HTTP2-TLS13] Benjamin, D., "Using TLS 1.3 with HTTP/2", RFC 8740, DOI 10.17487/RFC8740, February 2020, <https://www.rfc-editor.org/rfc/rfc8740>.

[QUIC] Iyengar, J., Ed. and M. Thomson, Ed., "QUIC: A UDP-Based Multiplexed and Secure Transport", RFC 9000, DOI 10.17487/RFC9000, May 2021, <https://www.rfc-editor.org/rfc/rfc9000>.

[QUIC-TLS] Thomson, M., Ed. and S. Turner, Ed., "Using TLS to Secure QUIC", RFC 9001, DOI 10.17487/RFC9001, May 2021, <https://www.rfc-editor.org/rfc/rfc9001>.

Acknowledgments

TODO acknowledge.

Authors' Addresses

Yaroslav Rosomakho
Zscaler
Email: yrosomakho@zscaler.com

Tirumaleswar Reddy
Nokia
Bangalore
Karnataka
India
Email: kondtir@gmail.com