Transport Layer Security
Internet-Draft
Intended status: Standards Track
Expires: 9 August 2026

R. Housley
Vigil Security

5 February 2026

Using the MLS Handshake in TLS 1.3
draft-housley-tls-using-mls-handshake-00

Abstract

This document specifies an extension to the Transport Layer Security (TLS) Protocol Version 1.3 that allows TLS clients and servers to use the Message Layer Security (MLS) Protocol handshake to establish the shared secret instead to the traditional shared secret establishment mechanism. The MLS protocol provides straightforward mechanism to update the shared secret that can be initiated by either the client or the server during the TLS session, and each epoch provides forward security and post-compromise security.

About This Document

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

The latest revision of this draft can be found at TBD. Status information for this document may be found at https://datatracker.ietf.org/doc/draft-housley-tls-using-mls-handshake/.

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/russhousley/draft-housley-tls-using-mls-handshake.

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/.

Copyright Notice

Table of Contents

1. Introduction

This document specifies an extension to the Transport Layer Security (TLS) Protocol Version 1.3 [RFC9846] that allows TLS clients and servers to use the Message Layer Security (MLS) Protocol [RFC9420] handshake to establish the shared secret. The resulting shared secret is used in the TLS key schedule to derive all of the cryptographic keying material for the TLS 1.3 protocol. The MLS

protocol provides a straightforward mechanism to update the shared secret, and either the client or the server can initiate the update during the TLS session. The period of time during which one shared secret is used is called an epoch. An epoch provides forward security and post-compromise security.

2. Terminology

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. Motivation and Design Rationale

There are operational motivations and security motivations for using the MLS handshake to manage TLS shared secrets.

An operational motivation is the straightforward mechanism for shared secret update that can be initiated by either the client or the server. The initial shared secret is based on the client KeyPackage and the server Welcome, which are carried in the TLS 1.3 ClientHello and the ServerHello, respectively. The KeyPackage and Welcome are specified in [RFC9420]. At any time during the TLS session, the client or server can update the keying material in their LeafNode of the MLS ratchet tree, which results in the computation of a fresh epoch, including a fresh TLS shared secret. By discarding all stale secret values, each epoch provides forward security and post-compromise security.

To apply the MLS protocol in TLS 1.3, the client provide a KeyPackage. In MLS, the KeyPackages are distributed by the Delivery Service (DS), but in the TLS environment, the the client KeyPackage is sent directly to the server in an extension in the ClientHello message. Once the TLS session is established, the client or server can provide an update to the keying material in their LeafNode in a mls_handshake message as specified in Section 6. Each KeyPackage is intended to be used only once. Reuse of KeyPackages can lead to replay attacks.

A security motivation is the ability to obtain both forward secrecy and post-compromise security, especially for long-lived TLS sessions. Forward secrecy means that TLS records sent at a certain point in time are secure in the face of later compromise of a the peer. Post-compromise security means that TLS records are secure even if the communicating peer was compromised at some point in the past.

Forward secrecy between epochs is provided by deleting private keys from past versions of the MLS ratchet tree, as this prevents old session secrets from being re-derived. Forward secrecy within an epoch is possible with the MLS protocol, but that feature is not currently supported in TLS.

Post-compromise security (PCS) is provided between epochs by the client or server regularly updating their leaf key in the MLS ratchet tree. Updating their leaf key prevents shared secrets from continuing to be encrypted to public keys whose private keys had previously been compromised.

Note that a client or server sending an update of the keying material for their LeafNode does not achieve PCS until the peer processes the MLS Commit. That is, the PCS guarantees come into effect when the peer processes the relevant Commit, not when the sender creates it.

This specification makes use of the two-party profile for MLS that is defined in [I-D.kohbrok-mls-two-party-profile].

4. Extension Overview

This section provides a brief overview of the "tls_using_mls_handshake" extension.

The client includes the "tls_using_mls_handshake" extension in the ClientHello message. The "tls_using_mls_handshake" extension MUST contain the client KeyPackage.

If the client includes both the "tls_using_mls_handshake" extension and the "early_data" extension, then the server MUST terminate the connection with an "illegal_parameter" alert.

If the server is willing to use MLS to establish the shared secret, then the server includes the "tls_using_mls_handshake" extension in the ServerHello message. The "tls_using_mls_handshake" extension MUST contain the Welcome message created by the server using the client KeyPackage.

When the "tls_using_mls_handshake" extension is successfully negotiated, the TLS 1.3 key schedule processing depends on the shared secret produced by the MLS ratchet tree and MLS key schedule.

The server MUST validate the client KeyPackage. KeyPackage validation depends on the GroupContext object, which captures MLS state:

```
struct {
    ProtocolVersion version = mls10;
    CipherSuite cipher_suite;
    opaque group_id<V>;
    uint64 epoch;
    opaque tree_hash<V>;
    opaque confirmed_transcript_hash<V>;
    Extension extensions<V>;
} GroupContext;
```

The fields in the MLS state have the following semantics for the TLS session:

* The cipher_suite is the cipher suite from the client KeyPackage, and the same cipher_suite MUST be used in the Welcome generated by the server.

* The group_id field is an application-defined identifier; it MUST contain "tls13".

* The epoch field represents the current version, starting with a value of 0x0000000000000000.

* The tree_hash field contains a commitment to the contents of the group’s ratchet tree and the credentials for the members of the group, which are always only the TLS client and the TLS server. See Section 7.8 of [RFC9420] for details.

* The confirmed_transcript_hash field contains a running hash over the MLS messages that led to this state. See Section 8.2 of [RFC9420] for details.

* The extensions field contains the details of any MLS protocol extensions that apply to the group. See Section 12.1.7 of [RFC9420] for details.

The server generates Welcome message, which MUST include an extension of type ratchet_tree to ensure that the client and the server have the same MLS ratchet tree. In MLS the ratchet tree is often provided by the Delivery Service, which is not an option in TLS. The Welcome message provides the client with the state of the group after it is created by the server:

```
struct {
  CipherSuite cipher_suite;
  EncryptedGroupSecrets secrets<V>;
  opaque encrypted_group_info<V>;
} Welcome;
```

See Section 12.4.3.1 of [RFC9420] for details regarding creations and processing of the Welcome message.

5. The "tls_using_mls_handshake" Extension

This section specifies the "tls_using_mls_handshake" extension, which MAY appear in the ClientHello message and ServerHello message. It MUST NOT appear in any other messages. The "tls_using_mls_handshake" extension MUST NOT appear in the ServerHello message unless the "tls_using_mls_handshake" extension appeared in the preceding ClientHello message. If an implementation recognizes the "tls_using_mls_handshake" extension and receives it in any other message, then the implementation MUST abort the handshake with an "illegal_parameter" alert.

The general extension mechanisms enable clients and servers to negotiate the use of specific extensions. Clients request extended functionality from servers with the extensions field in the ClientHello message. If the server responds with a HelloRetryRequest message, then the client sends another ClientHello message as described in Section 4.1.2 of [RFC9846], including the same "tls_using_mls_handshake" extension as the original ClientHello message, or aborts the handshake.

Many server extensions are carried in the EncryptedExtensions message; however, the "tls_using_mls_handshake" extension is carried in the ServerHello message. The "tls_using_mls_handshake" extension is only present in the ServerHello message if the server recognizes the "tls_using_mls_handshake" extension and the server is willing to use MLS key management and authentication in lieu of the traditional TLS 1.3 key management and authentication.

The Extension structure is defined in [RFC9846]; it is repeated here for convenience.

```
struct {
  ExtensionType extension_type;
  opaque extension_data<0..2^16-1>;
} Extension;
```

The "extension_type" identifies the particular extension type, and the "extension_data" contains information specific to the particular extension type.

This document specifies the "tls_using_mls_handshake" extension, adding one new type to ExtensionType:

```
enum {
 tls_using_mls_handshake(TBD0),
 (65535)
} ExtensionType;
```

The "tls_using_mls_handshake" extension is relevant when the client and server are willing to use the MLS protocol for key management and authentication. The "tls_using_mls_handshake" extension provides the client KeyPackage and the server KeyPackage needed to build the MLS ratchet tress for the client and server. The KeyPackage structure is defined in Section 10 of [RFC9420]. The "tls_using_mls_handshake" extension has the following syntax:

```
struct {
  select (Handshake.msg_type) {
      case client_hello: KeyPackage;
      case server_hello: Welcome;
  };
} TLSUsingMLSHandshake;
```

The TLS client creates its own KeyPackage for inclusion in the ClientHello. The KeyPackage includes a cipher suite and any MLS extensions that the client desires.

If the TLS server supports MLS handshake, then the TLS server inspects KeyPackage from the ClientHello to determine whether the cipher suite is supported and acceptable, whether all of the client provided MLS extensions are supported and acceptable, and whether the client credential is valid. If all of these checks are successful, then the TLS server creates the MLS group and sends the Welcome message in the ServerHello.

6. MLS Handshake Messages

If the "tls_using_mls_handshake" extension is successfully negotiated, then the client and the server can update the keying material in their LeafNode by encapsulating inside a TLS handshake message with a "mls_handshake" type. Only the MLS Commit message is needed to achieve this capability, but any MLS handshake message could be carried in this TLS handshake message to ensure future compatibility.

```
enum {
  mls_handshake(TBD1),
  (255)
} HandshakeType;
```

In the TLS protocol, the "mls_handshake" message encapsulates one of the MLS Handshake messages specified in [I-D.kohbrok-mls-two-party-profile] for key update or resumption. When updating the keying material in the MLS LeafNode, the Commit message MUST include the update by value. The TwoPartyMLSMessage types are:

```
struct {
  ProtocolVersion version = mls10;
  uint16 tpmlsmt;
  select (TwoPartyMLSMessage.tpmlsmt) {
      case mls_connection_update:
          Commit commit;
      case mls_epoch_key_update:
          uint64 epoch;
      case mls_resumption_request:
          Commit commit;
      case mls_resumption_response:
          Commit commit;
  };
} TwoPartyMLSMessage;
```

The format for each message is defined in [RFC9420].

Since the TLS environment does not include a Delivery Service, the MLS Commit message MUST contain the proposal to be applied by value as specified in Section 12.4 of [RFC9420].

6.1. Establishing the Initial TLS Shared Secret with MLS

The TLS server creates the MLS group, add it adds the client with the KeyPackage provided in the ClientHello, which results in a group of two members. The server send the Welcome message to the client in the ServerHello. The Welcome can include any MLS extensions that are supported by the client KeyPackage.

Upon successful completion of the MLS handshake, the exporter_secret produced by the MLS key schedule is used to derive the TLS shared secret, this value is referred to as the "Handshake Secret" in [RFC9846]. The TLS shared secret is derived as:

```
MLS-Exporter(Label, Context, Length) =
    ExpandWithLabel(DeriveSecret(exporter_secret,
        "TLS shared secret"), "exported",
        Hash(Context), Length)
```

Note: Length is the size of the TLS shared secret to be input into the TLS key schedule.

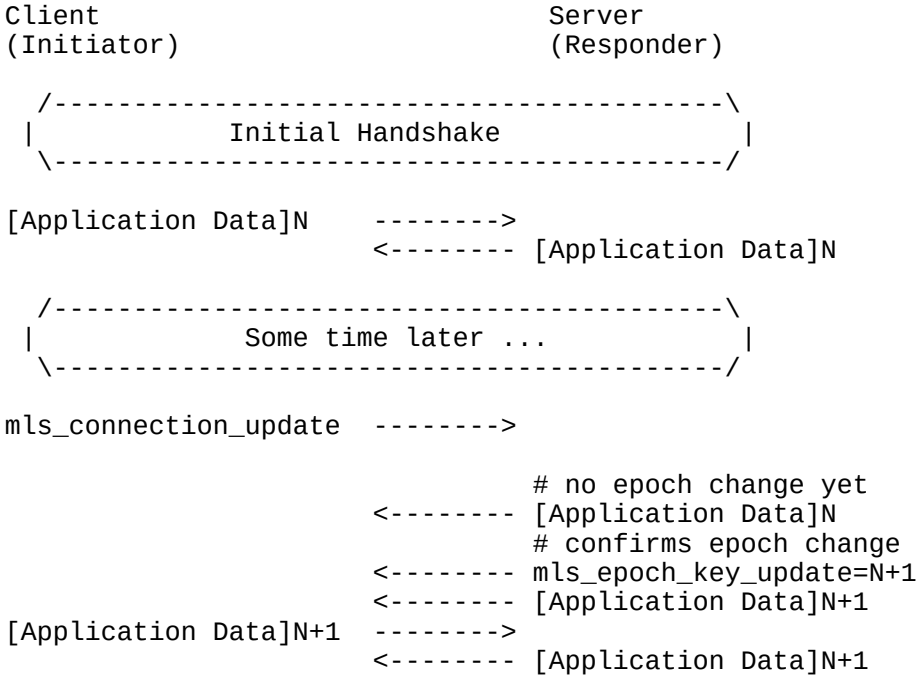
6.2. Updating the TLS Shared Secret with MLS

After the initial MLS handshake is successfully completed or a session is resumed, the client or the server can update the session key using a Commit message, which results in a new TLS shared secret. Once the new MLS key schedule is computed, a fresh "Handshake Secret" is exported to compute a new TLS key schedule.

To enforce the forward secrecy and post-compromise security, the client and the server periodically update the keys that represent them to the MLS group by sending a Commit message that includes an Update by value for their own LeafNode. See Section 12.1.2 of [RFC9420] for details.

Once the Commit message is sent, some traffic using the previous keying material might still be in flight. Once traffic using the new keying material is received, then the stale keying material can safely be discarded.

The following illustrates the update of the TLS shared secret.



Legend:

[]N Indicates messages protected with keys derived from epoch N

6.3. Resumption

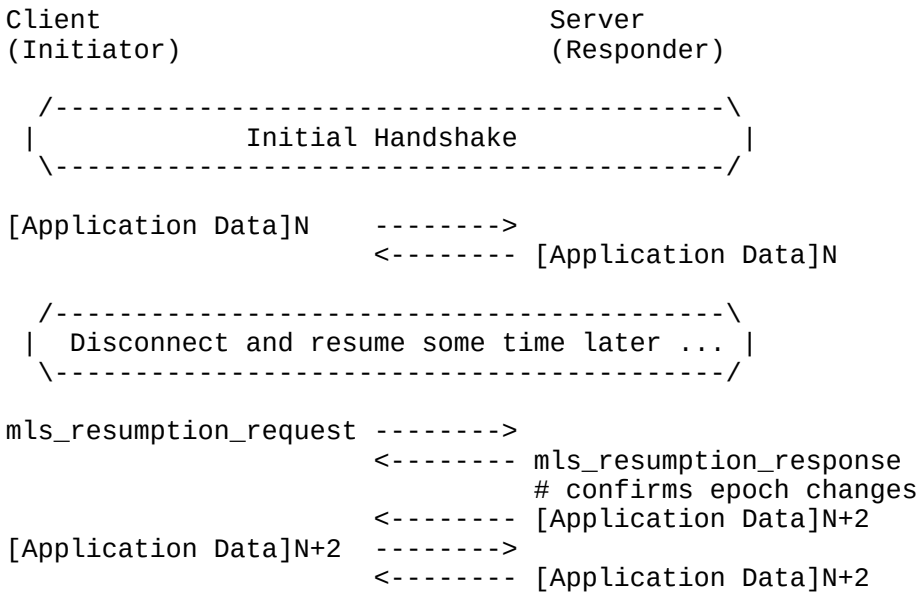
After the initial MLS handshake is successfully completed, a session can resume as long as the client and server have retained the MLS group state. The normal TLS 1.3 resumption process is described in Section 2.2 of [RFC9846].

To cryptographically separate the resumed session from the original session and ensure forward secrecy and post-compromise security, the client and the server each update their nodes in the MLS ratchet tree with a Commit message.

If the client had sent a Update before the TLS session was disconnected, and the client was waiting for a EpochKeyUpdate, then the client MUST use that pending Update message as the basis for the Commit message.

If the server had sent a Update before the TLS session was disconnected, and the server was waiting for a EpochKeyUpdate, then the server MUST use that pending Update message as the basis for the Commit message.

The following illustrates the update of the TLS shared secret after resumption.



Legend:

[]N Indicates messages protected with keys derived from epoch N

7. Security Considerations

The security considerations in [RFC9846] and [RFC9420] apply.

8. IANA Considerations

IANA is requested to update the "TLS ExtensionType Values" registry [IANA-TLS-EXT] to include "tls_using_mls_handshake" with a value of TBD0 and the list of messages "CH, SH" in which the "tls_using_mls_handshake" extension is allowed to appear.

IANA is requested to update the "TLS HandshakeType Values" registry [IANA-TLS-HS] to include "mls_handshake" with a value of TBD1, a DTLS-OK value of "Y", and a Reference to this document (once it is published as an RFC).

IANA is requested to update the "MLS Exporter Labels" registry [IANA-MLS-EL] to include "TLS shared secret", a Recommended value of "N", and a Reference to this document (once it is published as an RFC).

9. References

9.1. Normative References

[I-D.kohbrok-mls-two-party-profile]
Kohbrok, K. and R. Robert, "A two-party profile for MLS", Work in Progress, Internet-Draft, draft-kohbrok-mls-two-party-profile-00, 3 February 2026, <https://datatracker.ietf.org/doc/html/draft-kohbrok-mls-two-party-profile-00>.

[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>.

[RFC9420] Barnes, R., Beurdouche, B., Robert, R., Millican, J., Omara, E., and K. Cohn-Gordon, "The Messaging Layer Security (MLS) Protocol", RFC 9420, DOI 10.17487/RFC9420, July 2023, <https://www.rfc-editor.org/rfc/rfc9420>.

[RFC9846] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 9846, February 2026, <https://www.rfc-editor.org/rfc/rfc9846>.

9.2. Informative References

[IANA-MLS-EL]
IANA, "TMLS Exporter Labels", n.d., <https://www.iana.org/assignments/mls/mls.xhtml#mls-exporter-labels>.

[IANA-TLS-EXT]
IANA, "TLS ExtensionType Values", n.d., <https://www.iana.org/assignments/tls-extensiontype-values/tls-extensiontype-values.xhtml>.

[IANA-TLS-HS]
IANA, "TLS HandshakeType Values", n.d., <https://www.iana.org/assignments/tls-parameters/tls-parameters.xhtml#tls-parameters-7>.

Acknowledgments

Thanks to Sean Turner, Raphael Robert, and Konrad Kohbrok for providing constructive comments.

Author’s Address

Russ Housley
Vigil Security, LLC
Email: housley@vigilsec.com