Transport Layer Security
Internet-Draft
Intended status: Standards Track
Expires: 8 January 2026

H. Tschofenig
Siemens
M. T端xen
M端nster Univ. of Applied Sciences
T. Reddy
Nokia
S. Fries
Siemens
Y. Rosomakho
Zscaler

7 July 2025

# Extended Key Update for Transport Layer Security (TLS) 1.3
draft-ietf-tls-extended-key-update-05

## Abstract

TLS 1.3 ensures forward secrecy by performing an ephemeral Diffie-Hellman key exchange during the initial handshake, protecting past communications even if a party's long-term keys are later compromised. While the built-in KeyUpdate mechanism allows traffic keys to be refreshed during a session, it does not introduce new forward-secret key material. This limitation can pose a security risk in long-lived sessions, such as those found in industrial IoT or telecommunications environments.

To address this, this specification defines an extended key update mechanism that performs a fresh Diffie-Hellman exchange within an active session, thereby re-establishing forward secrecy beyond the initial handshake. By forcing attackers to exfiltrate new key material repeatedly, this approach mitigates the risks associated with static key compromise. Regular renewal of session keys helps contain the impact of such compromises. The extension is applicable to both TLS 1.3 and DTLS 1.3.

## 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

The Transport Layer Security (TLS) 1.3 protocol provides forward secrecy by using an ephemeral Diffie-Hellman (DHE) key exchange during the initial handshake. This ensures that encrypted communication remains confidential even if an attacker later obtains a party's long-term private key, protecting against passive adversaries who record encrypted traffic for later decryption.

TLS 1.3 also includes a KeyUpdate mechanism that allows traffic keys to be refreshed during an established session. However, this mechanism does not introduce new forward-secret key material, as it applies only a key derivation function to the previous application traffic secret as input. While this design is generally sufficient for short-lived connections, it may present security limitations in scenarios where sessions persist for extended periods, such as in industrial IoT or telecommunications systems, where continuous availability is critical and session renegotiation or resumption is impractical.

Earlier versions of TLS supported session renegotiation, which allowed peers to negotiate fresh keying material, including performing new Diffie-Hellman exchanges during the session lifetime. Due to protocol complexity and known vulnerabilities, renegotiation was first restricted by [RFC5746] and ultimately removed in TLS 1.3. While the KeyUpdate message was introduced to offer limited rekeying functionality, it does not fulfill the same cryptographic role as renegotiation and cannot refresh long-term secrets or derive new secrets from fresh DHE input.

Security guidance from national agencies, such as ANSSI (France), recommends the periodic renewal of cryptographic keys during long-lived sessions to limit the impact of key compromise. This approach encourage designs designs that force an attacker to perform dynamic key exfiltration, as defined in [RFC7624]. Dynamic key exfiltration refers to attack scenarios where an adversary must repeatedly extract fresh keying material to maintain access to protected data, increasing operational cost and risk for the attacker. In contrast, static key exfiltration, where a long-term secret is extracted once and reused, poses a greater long-term threat, especially when session keys are not refreshed with forward-secret input.

This specification defines a TLS extension that introduces an extended key update mechanism. Unlike the standard key update, this mechanism allows peers to perform a fresh Diffie-Hellman exchange within an active session using one of the groups negotiated during the initial handshake. By periodically rerunning (EC)DHE, this extension enables the derivation of new traffic secrets that are

independent of prior key material. As noted in Appendix F of [I-D.ietf-tls-rfc8446bis], this approach mitigates the risk of static key exfiltration and shifts the attacker burden toward dynamic key exfiltration.

The proposed extension is applicable to both TLS 1.3 and DTLS 1.3. For clarity, the term "TLS" is used throughout this document to refer to both protocols unless otherwise specified.

## 2. Terminology and Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

To distinguish the key update procedure defined in [I-D.ietf-tls-rfc8446bis] from the key update procedure specified in this document, we use the terms "standard key update" and "extended key update", respectively.

## 3. Negotiating the Extended Key Update

Client and servers use the TLS flags extension [I-D.ietf-tls-tlsflags] to indicate support for the functionality defined in this document. We call this the "extended_key_update" extension and the corresponding flag is called "Extended_Key_Update" flag.

The "Extended_Key_Update" flag proposed by the client in the ClientHello (CH) MUST be acknowledged in the EncryptedExtensions (EE), if the server also supports the functionality defined in this document and is configured to use it.

If the "Extended_Key_Update" flag is not set, servers ignore any of the functionality specified in this document and applications that require perfect forward security will have to initiate a full handshake.

## 4. Extended Key Update Messages

======= If the client and server agree to use the extended key update mechanism, the standard key update MUST NOT be used. In this case, the extended key update fully replaces the standard key update functionality.

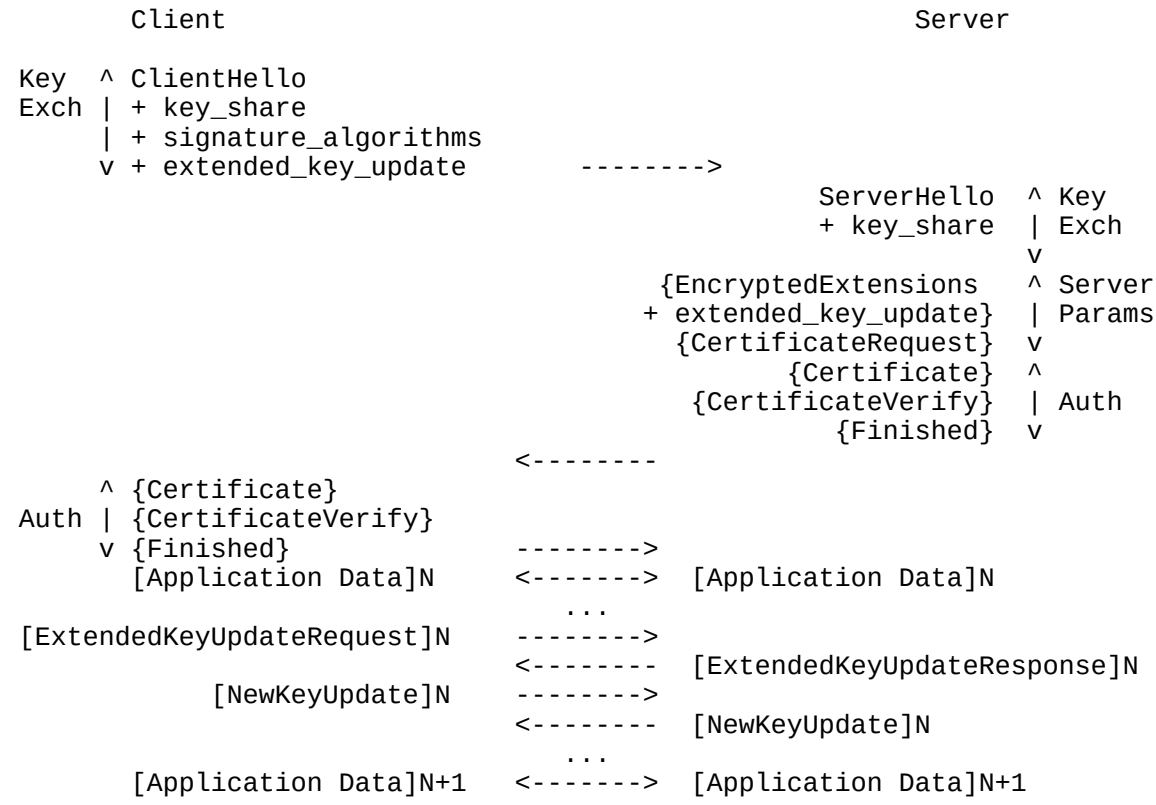
Implementations that receive a classic KeyUpdate message after successfully negotiating the Extended Key Update functionality MUST terminate the connection with an "unexpected_message" alert.

The extended key update handshake message exchange used to perform an update of cryptographic keys. This key update process can be sent by either peer after it has sent a Finished message. Implementations that receive a ExtendedKeyUpdateRequest message prior to receiving a Finished message MUST terminate the connection with an "unexpected_message" alert.

The KeyShareEntry in the ExtendedKeyUpdateRequest message and in the ExtendedKeyUpdateResponse message MUST be the same algorithm mutually supported by the client and server during the initial handshake. An implementation that receives an algorithm not previously negotiated MUST terminate the connection with an "illegal_parameter" alert.

Figure 1 shows the interaction graphically. First, support for the functionality in this specification is negotiated in the ClientHello and the EncryptedExtensions messages. Then, the ExtendedKeyUpdate exchange is sent to update the application traffic secrets.

The extended key update exchange is performed between the initiator and the responder whereby the initiator may be the TLS client or the TLS server.




Legend:

Indicates noteworthy extensions sent in the previously noted message.

\- Indicates optional or situation-dependent messages/extensions that are not always sent.

() Indicates messages protected using keys derived from a client_early_traffic_secret.

{} Indicates messages protected using keys derived from a [sender]_handshake_traffic_secret.

[]N Indicates messages protected using keys derived from [sender]_application_traffic_secret_N.

Figure 1: Extended Key Update Message Exchange.

The structure of the ExtendedKeyUpdate message is shown below.

```
struct {
  KeyShareEntry key_share;
} ExtendedKeyUpdateRequest;

enum {
  accepted(0),
  retry(1),
  rejected(2),
  clashed(3),
  (255)
} ExtendedKeyUpdateResponseStatus;

struct {
  ExtendedKeyUpdateResponseStatus status;
  select (ExtendedKeyUpdateResponse.status) {
     case accepted: KeyShareEntry key_share;
     case retry: uint8 delay;
  }
} ExtendedKeyUpdateResponse;

struct {
} NewKeyUpdate;
```

key_share: Key share information. The contents of this field are determined by the specified group and its corresponding definition. The structures are defined in [I-D.ietf-tls-rfc8446bis].

status: Response to ExtendedKeyUpdateRequest. This status field indicates whether responder accepted or declined Extended Key Update Request.

delay: Delay in seconds for the initiator to retry the request.

There are three rejection reasons defined:

1. retry: request was declined temporarily as responder is too busy. In this case ExtendedKeyUpdateResponse contains delay in seconds for initiator to retry. Initiator MUST NOT retry within this interval and SHOULD retry after it lapsed. Note that responder MAY apply an overall rate limit to extended key update that would not be specific to given TLS session. If initiator cannot proceed without immediate Extended Key Update it MUST terminate the connection with TLS alert "extended_key_update_required" (alert number TBD).

2. rejected: request was declined permanently. Initiator MUST NOT retry and if it cannot proceed without Extended Key Update it MUST terminate the connection with alert "extended_key_update_required" (alert number TBD).

3. clashed: request was declined because responder already initiated its own extended key update.

The exchange has the following steps:

1. Initiator sends a ExtendedKeyUpdateRequest message, which contains a key share. While an extended key update is in progress, the initiator MUST NOT initiate further key updates.

2. On receipt of the ExtendedKeyUpdateRequest message, the responder sends the ExtendedKeyUpdateResponse message. If the responder accepts the request, it sets the status to accepted and includes its own key share. If the responder declines the request, it sets the status accordingly and does not include the key share. While an extended key update is in progress, the responder MUST NOT initiate further key updates.

3. On receipt of the ExtendedKeyUpdateResponse message with accepted status, the initiator is able to derive a secret key based on the exchanged key shares. The NewKeyUpdate message is intentionally an empty structure that triggers the transition to new keying material.

4. On receipt of the NewKeyUpdate message by the responder, it MUST update its receive keys. In response, the responder transmits a NewKeyUpdate message and MUST update its sending keys.

5. After receiving the NewKeyUpdate message from the responder, the initiator MUST update its traffic keys and MUST send all its traffic using the next generation of keys.

Both sender and receiver MUST encrypt their NewKeyUpdate messages with the old keys. Both sides MUST ensure that the NewKeyUpdate encrypted with the old key is received before accepting any messages encrypted with the new key.

If TLS peers independently initiate the extended key update procedure and the requests cross in flight, the ExtendedKeyUpdateRequest message with the lower lexicographic order for the key_exchange value in the KeyShareEntry will be rejected by the responder using clashed status in ExtendedKeyUpdateResponse message. This approach prevents each side incrementing keys by two generations.

```
struct {
    HandshakeType msg_type;    /* handshake type */
    uint24 length;             /* bytes in message */
    select (Handshake.msg_type) {
        case client_hello:          ClientHello;
        case server_hello:          ServerHello;
        case end_of_early_data:     EndOfEarlyData;
        case encrypted_extensions:  EncryptedExtensions;
        case certificate_request:   CertificateRequest;
        case certificate:           Certificate;
        case certificate_verify:    CertificateVerify;
        case finished:              Finished;
        case new_session_ticket:    NewSessionTicket;
        case key_update:            KeyUpdate;
        case extended_key_update:   ExtendedKeyUpdate;
    };
} Handshake;
```

Figure 2: Handshake Structure.

## 5. Updating Traffic Secrets

When the extended key update message exchange is completed both peers have successfully updated their application traffic secrets. The key derivation function described in this document is used to perform this update.

The design of the key derivation function for computing the next generation of application_traffic_secret is motivated by the desire to include

* a secret derived from the (EC)DHE exchange (or from the hybrid key exchange / PQ-KEM exchange),

* a secret that allows the new key exchange to be cryptographally bind the previously established secret to the newly derived secret,

* the concatenation of the ExtendedKeyUpdateRequest and the ExtendedKeyUpdateResponse messages, which contain the key shares, binding the encapsulated shared secret ciphertext to IKM in case of hybrid key exchange, providing MAL-BIND-K-CT security (see [CDM23]), and

* new label strings to distinguish it from the key derivation used in TLS 1.3.

The following diagram shows the key derivation hierarchy.


During the initial handshake the Master Secret is generated, see Section 7.1 of [I-D.ietf-tls-rfc8446bis]. Since the Master Secret is discarded during the key derivation procedure, a derived value is stored. This value then serves as input to another key derivation step that takes the (EC)DHE-established value as a second parameter into account.

The traffic keys are re-derived from client_application_traffic_secret_N+1 and server_application_traffic_secret_N+1, as described in Section 7.3 of [I-D.ietf-tls-rfc8446bis].

Once client_/server_application_traffic_secret_N+1 and its associated traffic keys have been computed, implementations SHOULD delete client_/server_application_traffic_secret_N and its associated traffic keys as soon as possible. Note: The client_/server_application_traffic_secret_N and its associated traffic keys can only be deleted after receiving the NewKeyUpdate message.

When using this extension, it is important to consider its interaction with ticket-based session resumption. If resumption occurs without a new (EC)DH exchange that provides forward secrecy, an attacker could potentially revert the security context to an earlier state, thereby negating the benefits of the extended key update. To preserve the security guarantees provided by key updates, endpoints MUST either invalidate any session tickets issued prior to the key update or ensure that resumption always involves a fresh (EC)DH exchange.

If session tickets cannot be stored securely, developers SHOULD consider disabling ticket-based resumption in their deployments. While this approach may impact performance, it provides improved security properties.

## 6. Example

Figure 1 shows the interaction between a TLS 1.3 client and server graphically. This section shows an example message exchange where a client updates its sending keys.

There are two phases:

1. The support for the functionality in this specification is negotiated in the ClientHello and the EncryptedExtensions messages.

2. Once the initial handshake is completed, a key update can be triggered.

Figure 3 provides an overview of the exchange starting with the initial negotiation followed by the key update.

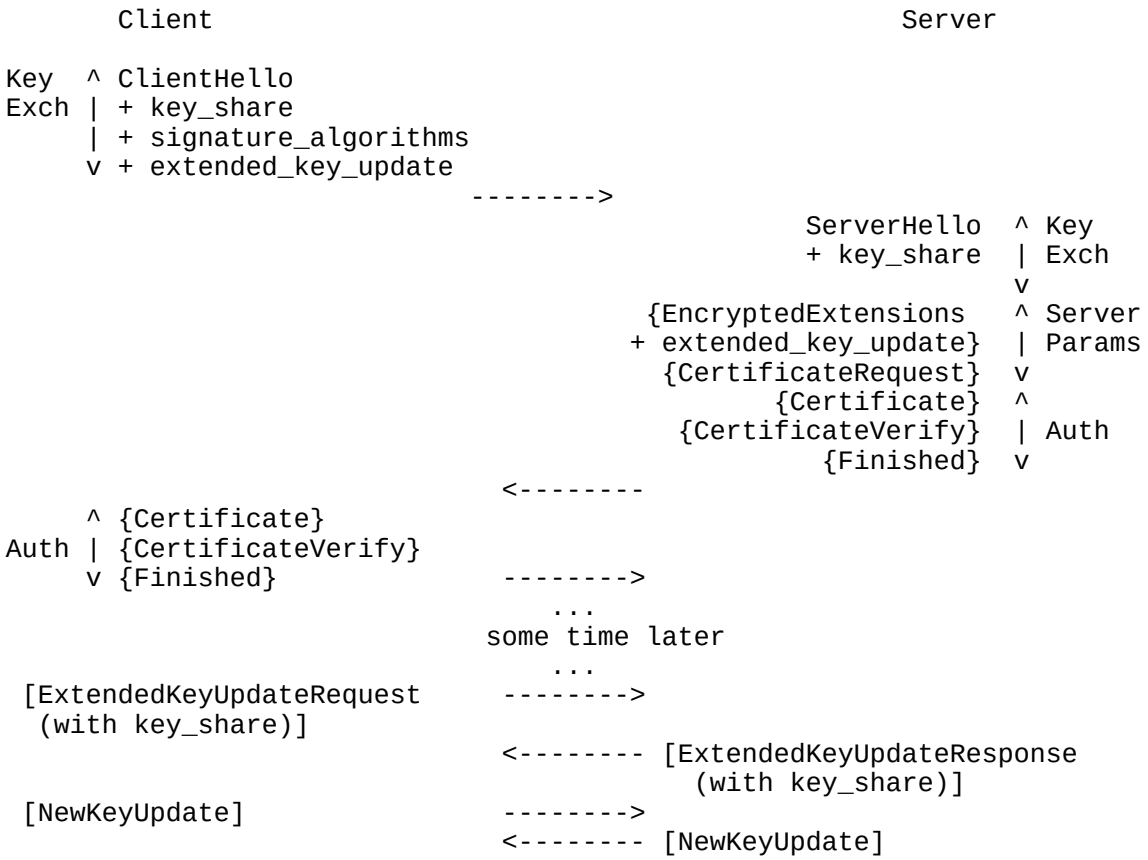



Figure 3: Extended Key Update Message Exchange.

## 7. DTLS 1.3 Considerations

Due to the possibility of a NewKeyUpdate message being lost and thereby preventing the sender of the NewKeyUpdate message from updating its keying material, receivers MUST retain the pre-update keying material until receipt and successful decryption of a message using the new keys.

Due to loss and/or reordering, DTLS 1.3 implementations may receive a record with an older epoch than the current one. They SHOULD attempt to process those records with that epoch but MAY opt to discard such out-of-epoch records.

## 8. Post-Quantum Cryptography Considerations

Hybrid key exchange refers to using multiple key exchange algorithms simultaneously and combining the result with the goal of providing security even if all but one of the component algorithms is broken. The transition to post-quantum cryptography motivates the introduction of hybrid key exchanges to TLS, as described in [I-D.ietf-tls-hybrid-design]. When the hybrid key exchange is used, then the key_exchange field of a KeyShareEntry in the initial exchange is the concatenation of the key_exchange field for each of the algorithms. The same approach is then re-used in the extended key update when key shares are exchanged.

## 9. SSLKEYLOGFILE Update

As a successful extended key update exchange invalidates previous secrets, SSLKEYLOGFILE [I-D.ietf-tls-keylogfile] needs to be populated with new entries. As a result, two additional secret labels are utilized in the SSLKEYLOGFILE:

1. CLIENT_TRAFFIC_SECRET_N+1: identifies the client_application_traffic_secret_N+1 in the key schedule

2. SERVER_TRAFFIC_SECRET_N+1: identifies the server_application_traffic_secret_N+1 in the key schedule

Similar to other entries in the SSLKEYLOGFILE, the label is followed by the 32-byte value of the Random field from the ClientHello message that established the TLS connection, and the corresponding secret encoded in hexadecimal.

SSLKEYLOGFILE entries for the extended key update MUST NOT be produced if SSLKEYLOGFILE was not used for other secrets in the handshake.

Note that each successful Extended Key Update invalidates all previous SSLKEYLOGFILE secrets including past iterations of CLIENT_TRAFFIC_SECRET_ and SERVER_TRAFFIC_SECRET_.

## 10. Exporter

Protocols like DTLS-SRTP and DTLS-over-SCTP utilize TLS or DTLS for key establishment but repurpose some of the keying material for their own purpose. These protocols use the TLS exporter defined in Section 7.5 of [I-D.ietf-tls-rfc8446bis].

Once the Extended Key Update mechanism is complete, such protocols would need to use the newly derived key to generate Exported Keying Material (EKM) to protect packets. The "sk" derived in the Section 5 will be used as the "Secret" in the exporter function, defined in Section 7.5 of [I-D.ietf-tls-rfc8446bis], to generate EKM, ensuring that the exported keying material is aligned with the updated security context.

## 11. Security Considerations

This entire document is about security.

## 12. IANA Considerations

### 12.1. TLS Alerts

IANA is requested to allocate value TBD for the "extended_key_update_required" alert in the "TLS Alerts" registry. The value for the "DTLS-OK" column is "Y".

### 12.2. TLS Flags

IANA is requested to add the following entry to the "TLS Flags" extension registry [TLS-Ext-Registry]:

* Value: TBD1
* Flag Name: extended_key_update
* Messages: CH, EE
* Recommended: Y
* Reference: [This document]

### 12.3. TLS HandshakeType

IANA is requested to add the following entries to the "TLS HandshakeType" registry [TLS-Ext-Registry].

#### 12.3.1. extended_key_update_request Message

* Value: TBD2
* Description: extended_key_update
* DTLS-OK: Y

* Reference: [This document]

* Comment:

#### 12.3.2. extended_key_update_response Message

* Value: TBD3

* Description: extended_key_update_response

* DTLS-OK: Y

* Reference: [This document]

* Comment:

#### 12.3.3. new_key_update Message

* Value: TBD3

* Description: new_key_update

* DTLS-OK: Y

* Reference: [This document]

* Comment:

## 13. References

### 13.1. Normative References

[I-D.ietf-tls-rfc8446bis]
Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", Work in Progress, Internet-Draft, draft-ietf-tls-rfc8446bis-12, 17 February 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-tls-rfc8446bis-12>.

[I-D.ietf-tls-tlsflags]
Nir, Y., "A Flags Extension for TLS 1.3", Work in Progress, Internet-Draft, draft-ietf-tls-tlsflags-15, 15 March 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-tls-tlsflags-15>.

[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>.

[RFC9147] 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>.

## 13.2. Informative References

[ANSSI-DAT-NT-003] ANSSI, "Recommendations for securing networks with IPsec, Technical Report", August 2015, <https://www.ssi.gouv.fr/uploads/2015/09/NT_IPsec_EN.pdf>.

[CDM23] ACM, "Keeping Up with the KEMs: Stronger Security Notions for KEMs and automated analysis of KEM-based protocols", November 2023, <https://eprint.iacr.org/2023/1933.pdf>.

[I-D.ietf-tls-hybrid-design] Stebila, D., Fluhrer, S., and S. Gueron, "Hybrid key exchange in TLS 1.3", Work in Progress, Internet-Draft, draft-ietf-tls-hybrid-design-13, 17 June 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-tls-hybrid-design-13>.

[I-D.ietf-tls-keylogfile] Thomson, M., Rosomakho, Y., and H. Tschofenig, "The SSLKEYLOGFILE Format for TLS", Work in Progress, Internet-Draft, draft-ietf-tls-keylogfile-05, 9 June 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-tls-keylogfile-05>.

[RFC5746] Rescorla, E., Ray, M., Dispensa, S., and N. Oskov, "Transport Layer Security (TLS) Renegotiation Indication Extension", RFC 5746, DOI 10.17487/RFC5746, February 2010, <https://www.rfc-editor.org/rfc/rfc5746>.

[RFC7296] Kaufman, C., Hoffman, P., Nir, Y., Eronen, P., and T. Kivinen, "Internet Key Exchange Protocol Version 2 (IKEv2)", STD 79, RFC 7296, DOI 10.17487/RFC7296, October 2014, <https://www.rfc-editor.org/rfc/rfc7296>.

[RFC7624] Barnes, R., Schneier, B., Jennings, C., Hardie, T., Trammell, B., Huitema, C., and D. Borkmann, "Confidentiality in the Face of Pervasive Surveillance: A

Threat Model and Problem Statement", RFC 7624, DOI 10.17487/RFC7624, August 2015, <https://www.rfc-editor.org/rfc/rfc7624>.

[RFC9325] Sheffer, Y., Saint-Andre, P., and T. Fossati, "Recommendations for Secure Use of Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)", BCP 195, RFC 9325, DOI 10.17487/RFC9325, November 2022, <https://www.rfc-editor.org/rfc/rfc9325>.

[TLS-Ext-Registry] IANA, "Transport Layer Security (TLS) Extensions", November 2023, <https://www.iana.org/assignments/tls-extensiontype-values>.

## Appendix A. Acknowledgments

We would like to thank the members of the "TSVWG DTLS for SCTP Requirements Design Team" for their discussion. The members, in no particular order, were:

* Marcelo Ricardo Leitner
* Zaheduzzaman Sarker
* Magnus Westerlund
* John Mattsson
* Claudio Porfiri
* Xin Long
* Michael T端xen
* Hannes Tschofenig
* K Tirumaleswar Reddy
* Bertrand Rault

Additionally, we would like to thank the chairs of the Transport and Services Working Group (tsvwg) Gorry Fairhurst and Marten Seemann as well as the responsible area director Martin Duke.

Finally, we would like to thank Martin Thomson, Ilari Liusvaara, Benjamin Kaduk, Scott Fluhrer, Dennis Jackson, David Benjamin, Matthijs van Duin, Rifaat Shekh-Yusef, Joe Birr-Pixton and Thom Wiggers for their review comments.

## Authors' Addresses

Hannes Tschofenig
Siemens
Email: hannes.tschofenig@gmx.net

Michael T端xen
M端nster Univ. of Applied Sciences
Email: tuexen@fh-muenster.de

Tirumaleswar Reddy
Nokia
Email: kondtir@gmail.com

Steffen Fries
Siemens
Email: steffen.fries@siemens.com

Yaroslav Rosomakho
Zscaler
Email: yrosomakho@zscaler.com