TLS
Internet-Draft
Intended status: Standards Track
Expires: 17 July 2026

Y. Sheffer
Intuit
I. Mihalcea
Y. Deshpande
Arm Limited
T. Fossati
Linaro
T. Reddy
Nokia

13 January 2026

Using Attestation in Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)
draft-fossati-seat-early-attestation-02

Abstract

The TLS handshake protocol allows authentication of one or both peers using static, long-term credentials. In some cases, it is also desirable to ensure that the peer runtime environment is in a secure state. Such an assurance can be achieved using remote attestation which is a process by which an entity produces Evidence about itself that another party can use to appraise whether that entity is found in a secure state. This document describes a series of protocol extensions to the TLS 1.3 handshake that enable the binding of the TLS authentication key to a remote attestation session. This enables an entity capable of producing attestation Evidence, such as a confidential workload running in a Trusted Execution Environment (TEE), or an IoT device that is trying to authenticate itself to a network access point, to present a more comprehensive set of security metrics to its peer. These extensions have been designed to allow the peers to use any attestation technology, in any remote attestation topology, and to use them mutually.

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://yaronf.github.io/draft-fossati-seat-early-attestation/draft-fossati-seat-early-attestation.html. Status information for this document may be found at https://datatracker.ietf.org/doc/draft-fossati-seat-early-attestation/.

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

Source for this draft and an issue tracker can be found at https://github.com/yaronf/draft-fossati-seat-early-attestation.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

Remote Attestation (RA) [RFC9334] is the process by which an entity produces evidence about itself that another party can use to evaluate the trustworthiness of that entity. This document describes a series of protocol extensions to the TLS 1.3 handshake that enable the binding of the TLS authentication key to a remote attestation session. This enables an attester, such as a confidential workload running in a Trusted Execution Environment (TEE) [I-D.ietf-teep-architecture], or an IoT device that is trying to authenticate itself to a network access point, to present a more comprehensive set of security metrics to its peer. This, in turn, allows for the implementation of authorization policies at the relying parties that are based on stronger security signals.

Given the variety of deployed and emerging attestation technologies (e.g., [TPM1.2], [TPM2.0], [I-D.ietf-rats-eat]) these extensions have been explicitly designed to be agnostic to the attestation formats. This is achieved by reusing the generic encapsulation defined in [I-D.ietf-rats-msg-wrap] for transporting Evidence and Attestation Results payloads in the TLS Attestation handshake message.

This specification provides both one-way (server-only) and mutual (client and server) authentication using traditional TLS authentication combined with attestation, and allows the attestation topologies at each peer to be independent of each other. The proposed design supports both background-check and passport topologies, as described in Sections 5.2 and 5.1 of [RFC9334]. This is detailed in Section 8.1 and Section 8.2.

The protocol we propose is implemented completely at the TLS level, resulting in several related advantages:

* Implementation is within a single system component.

* Security does not depend on application-level code, which tends to be less secure than widely shared infrastructure components.

* It is easier to reason about the application’s security, since the peers’ identities and security postures are known as soon as the handshake completes and the TLS connection is established.

* Application code does not need to change. At most, some configuration is needed, similar to the current use of certificate trust stores.

This document does not mandate any particular attestation technology.

2. Conventions and Terminology

The reader is assumed to be familiar with the vocabulary and concepts defined in Section 4 of [RFC9334].

The following terms are used in this document:

TLS Identity Key (TIK):
: A cryptographic key used by one of the peers to authenticate itself during the TLS handshake. The protocol's security is critically dependent on the provenance, lifetime and protection properties of the TIK. The TIK MUST be the X.509 certificate's end entity key and is maintained and protected by the TEE.

TIK-C, TIK-S:
: The TIK that identifies the client or the server, respectively.

TIK-C-ID, TIK-S-ID:
: An identifier for TIK-C or respectively, TIK-S. This may be a fingerprint (cryptographic hash) of the public key, but other implementations are possible.

Attestation binder:
: A cryptographic value used to bind the TLS handshake to the remote attestation session. May also be referred to as "binder" throughout the document.

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. Overview

The basic functional goal is to link the authenticated key exchange of TLS with an interleaved remote attestation session in such a way that the key used to sign the handshake can be proven to be residing within the boundaries of an attested TEE. The requirement is that the attester can provide Evidence containing the security status of both the signing key and the platform that is hosting it. The associated security goal is to obtain such binding so that no replay, relay or splicing from an adversary is possible.

The protocol's security relies on the verifiable binding between the TLS Identity Key, the specific TLS session and the platform state through attestation Evidence or Attestation Results conveyed in the CMW (Conceptual Message Wrapper) [I-D.ietf-rats-msg-wrap] payload.

3.1. Authentication vs. Attestation

The protocol combines platform attestation with X.509 certificate authentication.

Attestation when used alone is vulnerable to identity spoofing attacks, in particular when zero-day attacks exist for a class of hardware. (TODO: reference). Therefore it needs to be combined with traditional authentication, which in the case of TLS takes the form of CA-signed certificates.

We RECOMMEND that regular applications use authentication and attestation in tandem, to gain the full security guarantees of an authenticated TLS handshake (for the peer/peers being authenticated) as well as guarantees of platform integrity.

3.2. Integration into the TLS Handshake

The lightweight integration of attestation into the TLS handshake is designed to have minimal impact on the existing TLS security properties. The changes consist of:

* Negotiation extensions: New TLS extensions are added to ClientHello and EncryptedExtensions messages to negotiate the use of attestation and indicate supported attestation formats and verifiers.

* Independent handshake message: A new Attestation handshake message is introduced that carries attestation Evidence or Attestation Results. This message is completely independent of the standard TLS handshake flow and does not interfere with existing handshake messages or their processing.

* Independent key derivation: Key derivation for attestation (see Section 5.6) ensures independence of the regular TLS key schedule. As a result, attestation processing does not affect the standard TLS key derivation and security properties.

This minimal integration approach provides an intuitive explanation of why the addition of attestation does not adversely affect TLS security. The attestation components operate independently, leaving the core TLS handshake protocol and key derivation mechanisms unmodified. Nevertheless, formal validation of these security properties is still required.

4. Attestation Extensions

As typical with new features in TLS, the client indicates support for the new extension in the ClientHello message. The newly introduced extensions allow attestation Evidence or Attestation Results to be exchanged. Freshness of the exchanged Evidence is guaranteed through secret derivation from the TLS main secret and message transcript (see Section 5.6) when the Background Check Model is in use. In the Passport Model, freshness expectations are more relaxed and are governed by the lifetime of the signed Attestation Results.

When either the Evidence or the Attestation Results extension is successfully negotiated, attestation Evidence or Attestation Results are conveyed in an Attestation handshake message (see Section 4.1). The CMW payload in the Attestation message contains the attestation Evidence or Attestation Results encoded according to [I-D.ietf-rats-msg-wrap].

The attestation payload MUST contain assertions relating to the attester’s TLS Identity Key (TIK-C for client attester, TIK-S for server attester), which associate the private key with the attestation information. The TEE’s signature over the Evidence or AttestationResults within the CMW MUST include an attestation binder derived from the TLS main secret and the message transcript up to ServerHello (see Section 5.6) and the attester’s TLS identity public key, as specified in Section 4.1.

The relying party can obtain and appraise the remote Attestation Results either directly from the Attestation message (in the Passport Model), or by relaying the Evidence from the Attestation message to the Verifier and receiving the Attestation Results. Subsequently, the attested key is used to verify the CertificateVerify message, which remains unchanged from baseline TLS.

When using the Passport Model, the remote Attestation Results obtained by the attester from its trusted Verifiers can be cached and used for any number of subsequent TLS handshakes, as long as the freshness policy requirements are satisfied.

In TLS a client has to demonstrate possession of the private key via the CertificateVerify message, when client-based authentication is requested. This behavior remains unchanged in the current protocol, with the CertificateVerify message proving possession of the TIK.

This protocol supports both monolithic and split implementations. In a monolithic implementation, the TLS stack is completely embedded within the TEE. In a split implementation, the TLS stack is located outside the TEE, but any private keys (and in particular, the TIK)

only exist within the TEE. In order to support both options, only the TIK's identity, its public component and a short generated binder are ever passed between the Client or Server TLS stack and its Attestation Service. While the two types of implementations offer identical functionality, their security properties often differ, see Section 10.1 for more details.

4.1. Attestation Handshake Message

When attestation is negotiated via the extensions defined in this document, attestation Evidence or Attestation Results are conveyed in a new handshake message type: Attestation. This message carries a CMW (Conceptual Message Wrapper) payload as defined in [I-D.ietf-rats-msg-wrap].

The Attestation message structure is defined as follows:

```
enum {
    /* other handshake message types defined in {{I-D.ietf-tls-rfc8446bis}} */
    attestation(TBD),
    (255)
} HandshakeType;

struct {
    HandshakeType msg_type;    /* handshake type */
    uint24 length;             /* bytes in message */
    select (Handshake.msg_type) {
        case attestation:
            Attestation;
        /* other handshake message types */
    };
} Handshake;

struct {
    opaque cmw_payload<1..2^24-1>;
} Attestation;
```

Figure 1: Attestation Handshake Message Structure.

The cmw_payload field contains a CMW structure as defined in [I-D.ietf-rats-msg-wrap]. Both JSON and CBOR serializations are allowed in CMW, with the emitter choosing which serialization to use.

The CMW payload MUST contain attestation Evidence (in Background Check Model) or Attestation Results (in Passport Model) that binds the TLS Identity Key (TIK) to the platform and workload state. The TEE's signature over the Evidence or AttestationResults within the CMW MUST include:

* A binder derived from the TLS main secret and the message transcript, up to ServerHello, ensuring freshness of the attestation.

* The attester’s TLS identity public key (TIK-C for client attester, TIK-S for server attester)

This binding ensures that the attested key is the one used in the TLS handshake and provides freshness guarantees through secret derivation. See Section 5.6 for details.

5. Use of Attestation in the TLS Handshake

For both the Passport Model (described in section 5.1 of [RFC9334]) and Background Check Model (described in Section 5.2 of [RFC9334]) the following modes of operation are allowed when used with TLS, namely:

* TLS client is the attester,

* TLS server is the attester, and

* TLS client and server mutually attest towards each other.

We will show the message exchanges of the first two cases in sub-sections below. Mutual authentication via attestation combines these two (non-interfering) flows, including cases where one of the peers uses the Passport Model for its attestation, and the other uses the Background Check Model.

5.1. Handshake Overview

The handshake defined here is analogous to certificate-based authentication in a regular TLS handshake. The peer being attested first proves possession of the private key using the CertificateVerify message, which remains unchanged from baseline TLS. Following that, the TLS Identity Key (TIK) is bound by the TEE to the attestation credential being carried in a new Attestation handshake message (see Section 4.1).

The attestation Evidence or Attestation Results are conveyed in an Attestation handshake message (see Section 4.1), which carries a CMW payload as defined in [I-D.ietf-rats-msg-wrap].

5.2. TLS Client Authenticating Using Evidence

In this use case, the TLS server (acting as a relying party) challenges the TLS client (as the attester) to provide Evidence. A session-specific value is derived (see Section 5.6) which incorporates randomness from both client and server, and this value is fed into the generation of the Evidence. The client sends the Evidence in an Attestation handshake message after the CertificateVerify message. The TLS server, when receiving the Evidence, will have to contact the Verifier (which is not shown in the diagram).

An example of this flow can be found in device onboarding where the client initiates the communication with cloud infrastructure to get credentials, firmware and other configuration data provisioned to the device. For the server to consider the device genuine it needs to present Evidence.

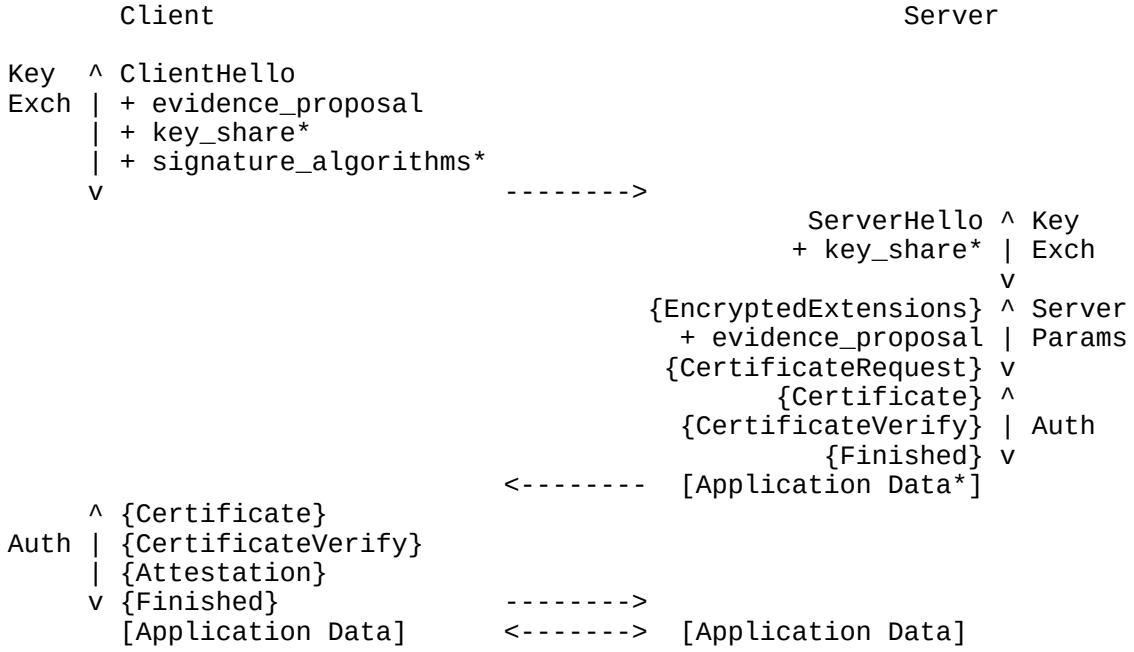

Figure 2: TLS Client Providing Evidence to TLS Server.

5.3. TLS Server Authenticating Using Evidence

In this use case the TLS client challenges the TLS server to present Evidence. The TLS server acts as an attester while the TLS client is the relying party. The server sends the Evidence in an Attestation handshake message after the CertificateVerify message. The TLS client, when receiving the Evidence, will have to contact the Verifier (which is not shown in the diagram).

An example of this flow can be found in confidential computing where a compute workload is only submitted to the server infrastructure once the client/user is assured that the confidential computing platform is genuine.

Figure 3: TLS Server Providing Evidence to TLS Client.

5.4. TLS Client Authenticating Using Attestation Results

In this use case the TLS client, as the attester, provides Attestation Results to the TLS server. The TLS client is the attester and the TLS server acts as a relying party. Prior to delivering its Certificate message, the client must contact the Verifier (not shown in the diagram) to receive the Attestation Results that it will use as credentials. The client sends the Attestation Results in an Attestation handshake message after the

CertificateVerify message.

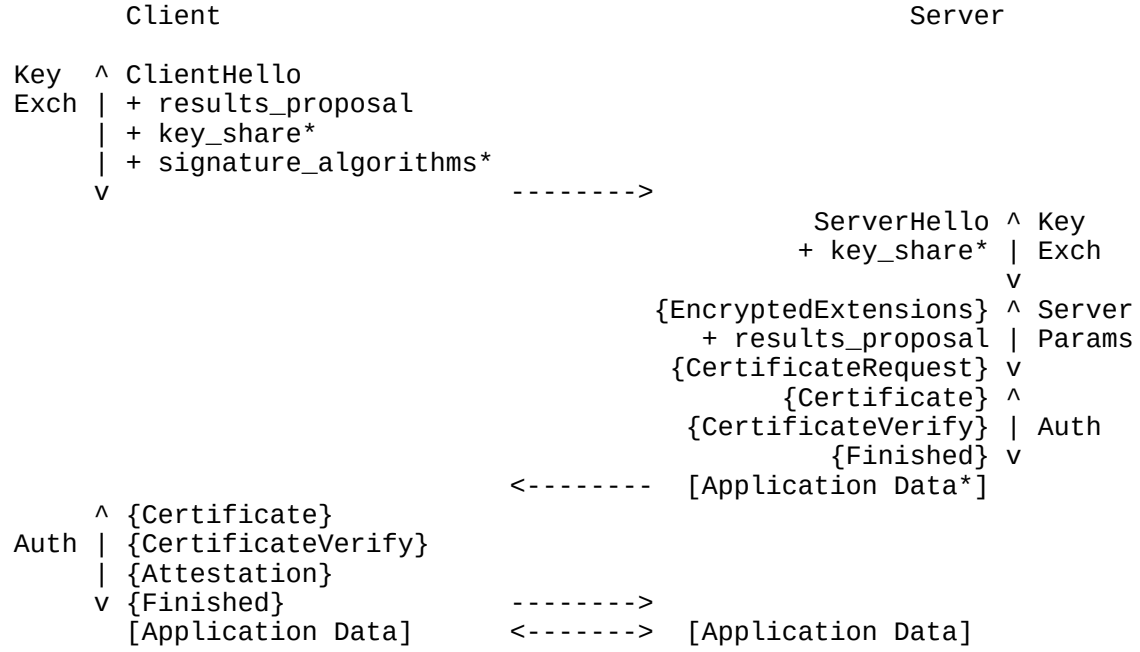

Figure 4: TLS Client Providing Results to TLS Server.

5.5. TLS Server Authenticating Using Attestation Results

In this use case the TLS client, as the relying party, requests Attestation Results from the TLS server. Prior to delivering its Certificate message, the server must contact the Verifier (not shown in the diagram) to receive the Attestation Results that it will use as credentials. The server sends the Attestation Results in an Attestation handshake message after the CertificateVerify message.

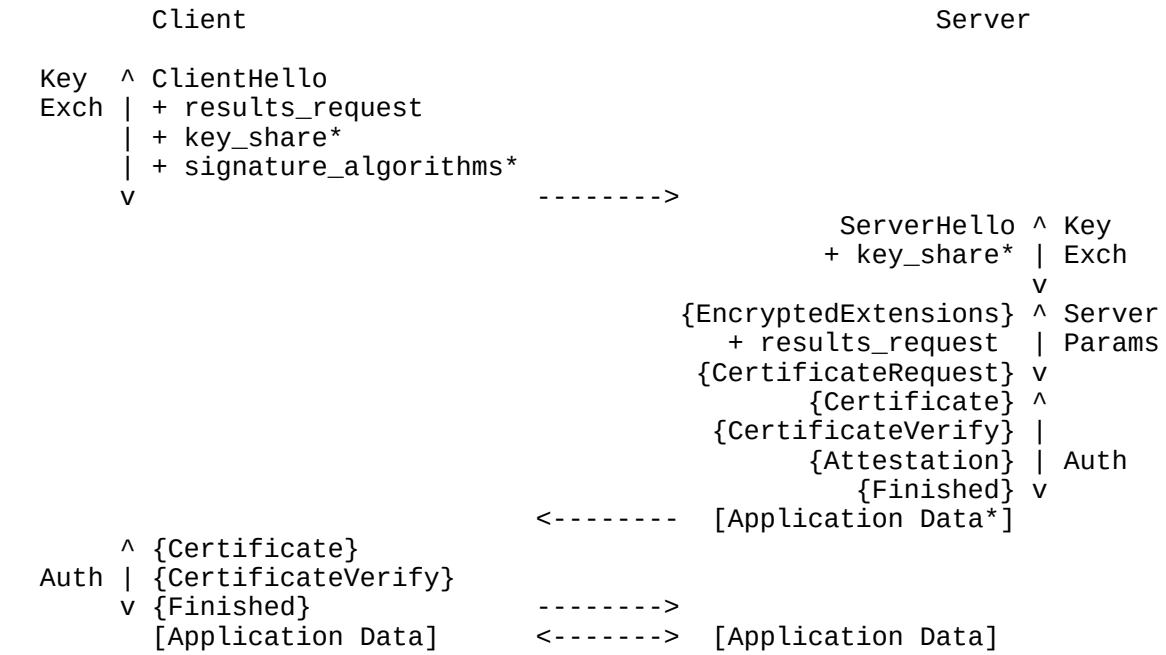

Figure 5: TLS Server Providing Attestation Results to TLS Client.

5.6. Cryptographic Operations

This section defines the key derivation for attestation, which operates independently from the regular TLS key schedule as described in Section 7.1 of [I-D.ietf-tls-rfc8446bis]. The attestation key derivation adds new secrets to the standard TLS key schedule without modifying any existing key derivation steps. The standard TLS key schedule is unchanged before and after Main Secret as specified in [I-D.ietf-tls-rfc8446bis].

The attestation key derivation uses HKDF Section 7.1 of [I-D.ietf-tls-rfc8446bis] to derive attestation-specific secrets from the TLS main secret. Two attestation main secrets are derived: one for the client (c_attest_main) and one for the server (s_attest_main).

The attestation key derivation adds the following branches to the standard TLS key schedule (Fig. 5 of [I-D.ietf-tls-rfc8446bis]):

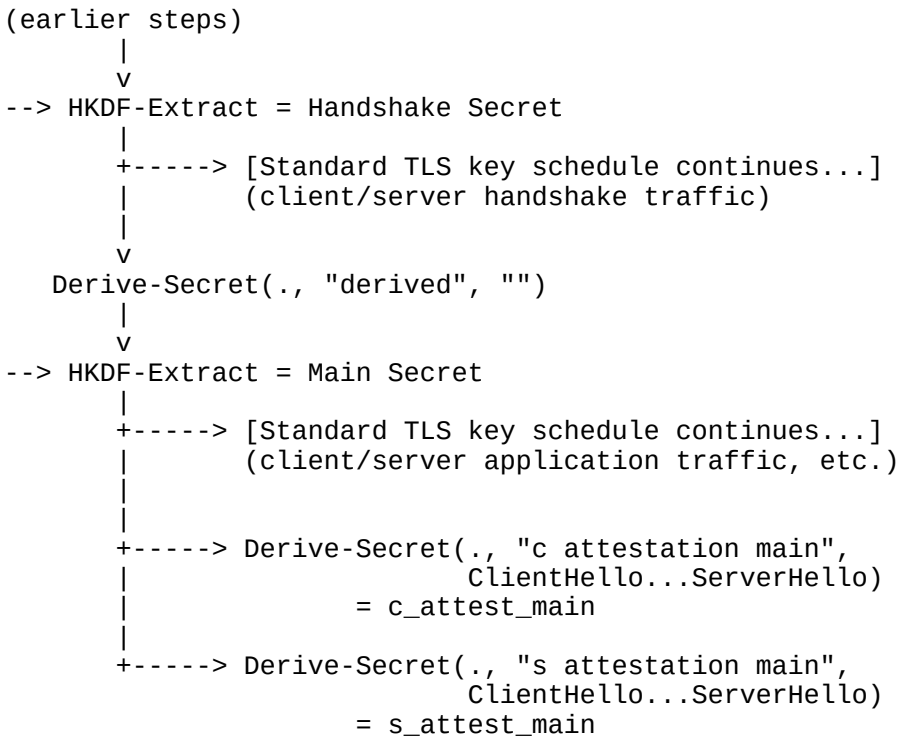

Figure 6: Attestation Key Schedule Additions.

The attestation main secrets (c_attest_main and s_attest_main) are derived from the TLS main secret using Derive-Secret as defined in Section 7.1 of [I-D.ietf-tls-rfc8446bis], with the labels "c attestation main" and "s attestation main" respectively, and the handshake transcript up to and including ServerHello as the context.

The client’s attestation binder (c_attest_binder) that will be signed by the TEE is derived by applying HKDF-Expand-Label to c_attest_main with the label "attestation" and the client’s TLS public key as the context:

```
c_attest_binder = HKDF-Expand-Label(c_attest_main, "attestation",
                                    TLS_Client_Public_Key, Hash.length)
```

Similarly, the server’s attestation binder (s_attest_binder) is derived from s_attest_main:

```
s_attest_binder = HKDF-Expand-Label(s_attest_main, "attestation",
                                    TLS_Server_Public_Key, Hash.length)
```

The attestation binder is derived independently by both the attester and the peer. The attester incorporates this attestation binder into the Evidence. Upon receipt of the Attestation handshake message, the peer will have to derive the expected attestation binder using the same inputs and verify that the computed attestation binder matches the one in the Evidence. If this verification fails, the peer will treat the attestation as invalid. This verification ensures that the Evidence is bound to the specific TLS session and TLS public key being attested.

5.7. Binding the TIK to the TEE

This specification assumes that the TIK private key corresponding to the end-entity certificate used in the TLS handshake is generated inside a TEE and never leaves it. A platform could instead generate the TIK private key outside the TEE and compute the CertificateVerify signature using that external key. A relying party cannot detect this attack unless additional safeguards are in place.

This risk is particularly relevant in split deployments, where the TLS stack does not reside inside the TEE. In such architectures, attesting the TEE alone does not prove that the TIK private key used by the TLS endpoint was generated, is stored, or is controlled by the TEE.

To address this, the Evidence MUST include the TIK public key (TIK_pub). The relying party MUST verify that the TIK_pub included in the Evidence matches the public key presented in the TLS Certificate message. This binds the attestation Evidence to the TLS identity used for authentication.

Without this binding, a non-TEE TLS endpoint can obtain Evidence from a separate TLS endpoint that genuinely runs inside a TEE and relay that Evidence to the relying party while executing the TLS handshake itself. If the Evidence only attests that a TLS stack is running in a TEE, the relying party cannot determine whether the attested TLS stack is the one that actually performed the handshake. Binding the Evidence to the TIK public key prevents this relay attack.

The proposed binding ensures that the relying party does not establish a TLS session with a TLS endpoint whose TIK is not generated and controlled by the TEE. It does not attempt to protect the confidentiality of the TLS main secret in split deployments, where the TLS stack executes in the rich OS and remains susceptible to compromise.

5.8. The TLS Stack’s Interface to the TEE

When the TEE signs the Evidence or Attestation Results, it also binds them to the TLS Identity public key and the TLS session. TEE implementations differ, and some only allow a single user-provided challenge value to be added to the Evidence with no associated checks. Therefore we adopt a defense-in-depth approach:

* Separate attesting applications within the same TEE SHOULD NOT be capable of impersonating each other via Evidence or Attestation Results. Therefore, if multiple applications are expected to use attestation credentials, evidence/AR generation APIs SHOULD reflect identifiers for the calling contexts into the generated credential. These identifiers can be reflected as separate claims in the credential, or can be measured as part of more generic claims. A Relying Party SHOULD be capable of differentiating between the attesting applications based on their credentials.

* The RP SHOULD NOT base its trust decision only on the Attester’s trust root. It SHOULD also ensure that the entire attested software stack is endorsed.

* The TEE itself, when possible, SHOULD generate the attestation secret by running the derivation operations defined in Section 5.6, and, if it holds the TIK, SHOULD validate the public key. The attestation secret can be generated by the TEE only if TLS is running inside the TEE.

6. DTLS Considerations

The Attestation message MUST be handled using the existing DTLS handshake mechanisms for fragmentation, ordering, and retransmission to ensure reliable delivery.

Note that Attestation messages typically exceed 1,500 bytes in size. This means that the message will be split into multiple DTLS records, increasing the latency of handshake completion. This is particularly the case over channels where reordering and loss are more common due to factors such as routing transients, intermittent connectivity or mobility.

In DTLS, handshake messages that do not solicit a response are acknowledged using the DTLS ACK message. Because the Attestation handshake message does not elicit a response, the receiving peer MUST send a DTLS ACK upon receipt of the Attestation message. This ACK confirms only that the message was received; it does not indicate that attestation appraisal has completed.

Once the attester receives the ACK, it MUST stop retransmitting the Attestation message. The receiving peer performs attestation appraisal asynchronously and applies its authorization policy once appraisal results become available.

7. After The Initial Handshake

This section covers protocol behavior after the initial handshake, including session resumption, reattestation and the interaction between them.

7.1. Session Resumption

TLS 1.3 supports session resumption using Pre-Shared Keys (PSK) as defined in Section 4.6 of [I-D.ietf-tls-rfc8446bis]. When using attestation, session resumption works normally when reattestation is not required.

If client reattestation is required according to local policy (e.g., based on timing since the last attestation or changes in attestation state), session resumption MUST be rejected. The decision to reject resumption is per local policy and may depend on the timing of the resumption attempt relative to the required reattestation period. When resumption is rejected, the client MUST initiate a full handshake with attestation to obtain fresh attestation Evidence or Attestation Results.

The rationale for rejecting resumption when reattestation is required is that attestation state may have changed since the original handshake, and fresh verification is needed to ensure the peer’s platform and workload remain in a trustworthy state. If the client wishes to retain a long-running connection, it SHOULD perform reattestation Section 7.2 periodically, as per local policy.

7.2. Reattestation

Over time, attestation Evidence or Attestation Results may become stale and require refresh. Long-lived TLS connections require updated assurance that the peer continues to operate in a trustworthy state. This document therefore supports reattestation, in which either peer MAY request fresh Evidence at any time post-handshake. The attester MUST generate evidence using a freshly derived attestation_binder.

Reattestation is tied to the completion of an Extended Key Update (EKU) exchange [I-D.ietf-tls-extended-key-update]. TLS peers that require reattestation MUST support EKU, since reattestation depends on the key schedule update defined in the EKU draft. The first two messages of an EKU exchange introduce fresh key-exchange input and make Main Secret N+1 available to both peers.

The Attestation message MUST be sent immediately before the attestor sends its EKU(new_key_update) message. Once Main Secret N+1 is available (after the first two EKU messages), the attester derives a new attestation_binder from Main Secret N+1, using the concatenation of the EKU request and response messages and its TLS identity public key as context.

The receiving peer, however, MUST NOT process the Attestation until the EKU exchange and the authenticated transition step have completed. This ensures that attestation bound to Main Secret N+1 is accepted only after both peers have confirmed that they share the same updated key state.

For a client attester:

```
client_attestation_binder =
      Derive-Secret(Main Secret N+1,
                    "reattestation",
                    EKU(request) ||
                    EKU(response) ||
                    TLS_Client_Public_Key)
```

For a server attester:

```
server_attestation_binder =
      Derive-Secret(Main Secret N+1,
                    "reattestation",
                    EKU(request) ||
                    EKU(response) ||
                    TLS_Server_Public_Key)
```

Including the EKU request and response messages ensures that the resulting attestation binder is bound to the specific EKU exchange and therefore reflects fresh key-exchange entropy introduced by EKU.

After deriving the fresh attestation_binder, the attester:

1. generates fresh Evidence using the new attestation_binder and

2. sends a new Attestation handshake message containing the updated CMW payload.

The TLS peer validates the attestation by deriving and verifying the attestation binder as specified in Section 5.6.

Reattestation uses the Attestation formats that were negotiated during the initial handshake, there is no re-negotiation at this stage.

The decision to initiate reattestation is per local policy and may be based on factors such as elapsed time since the last attestation, changes in platform state, or security policy requirements.

8. Negotiating This Protocol

This section defines the TLS extensions used to negotiate the use of attestation in the TLS handshake. Two models are supported: the Background Check Model, where Evidence is exchanged and verified during the handshake, and the Passport Model, where pre-verified Evidence in the form of Attestation Results are presented. The extensions defined here allow peers to indicate their support for attestation and negotiate which attestation format and Verifier to use.

// Can we simplify this structure: remove the dual request/proposal,
// and unify the evidence+AR to a single negotiation extension. But
// also express Passport mode with and without freshness.

8.1. Evidence Extensions (Background Check Model)

The EvidenceType structure contains an indicator for the type of Evidence expected in the Attestation handshake message. The Evidence contained in the CMW payload is sent in the Attestation handshake message (see Section 4.1).

```
enum { CONTENT_FORMAT(0), MEDIA_TYPE(1) } typeEncoding;

struct {
    typeEncoding type_encoding;
    select (EvidenceType.type_encoding) {
        case CONTENT_FORMAT:
            uint16 content_format;
        case MEDIA_TYPE:
            opaque media_type<0..2^16-1>;
    };
} EvidenceType;

struct {
    select(Handshake.msg_type) {
        case client_hello:
            EvidenceType supported_evidence_types<1..2^8-1>;
        case server_hello:
        case encrypted_extensions:
            EvidenceType selected_evidence_type;
    }
} evidenceRequestTypeExtension;

struct {
    select(Handshake.msg_type) {
        case client_hello:
            EvidenceType supported_evidence_types<1..2^8-1>;
        case server_hello:
        case encrypted_extensions:
            EvidenceType selected_evidence_type;
    }
} evidenceProposalTypeExtension;
```

Figure 7: TLS Extension Structure for Evidence.

Values for media_type are defined in [iana-media-types]. Values for content_format are defined in [iana-content-formats].

8.2. Attestation Results Extensions (Passport Model)

```
struct {
    opaque verifier_identity<0..2^16-1>;
} VerifierIdentityType;

struct {
    select(Handshake.msg_type) {
        case client_hello:
            VerifierIdentityType trusted_verifiers<1..2^8-1>;

        case server_hello:
        case encrypted_extensions:
            VerifierIdentityType selected_verifier;
    }
} resultsRequestTypeExtension;

struct {
    select(Handshake.msg_type) {
        case client_hello:
            VerifierIdentityType trusted_verifiers<1..2^8-1>;

        case server_hello:
        case encrypted_extensions:
            VerifierIdentityType selected_verifier;
    }
} resultsProposalTypeExtension;
```

Figure 8: TLS Extension Structure for Attestation Results.

In the Passport Model, Attestation Results are sent in an Attestation handshake message (see Section 4.1) containing a CMW structure. The CMW structure is defined in [I-D.ietf-rats-msg-wrap].

9. TLS Client and Server Handshake Behavior

The high-level message exchange in Figure 9 shows the evidence_proposal, evidence_request, results_proposal, and results_request extensions added to the ClientHello and the EncryptedExtensions messages.

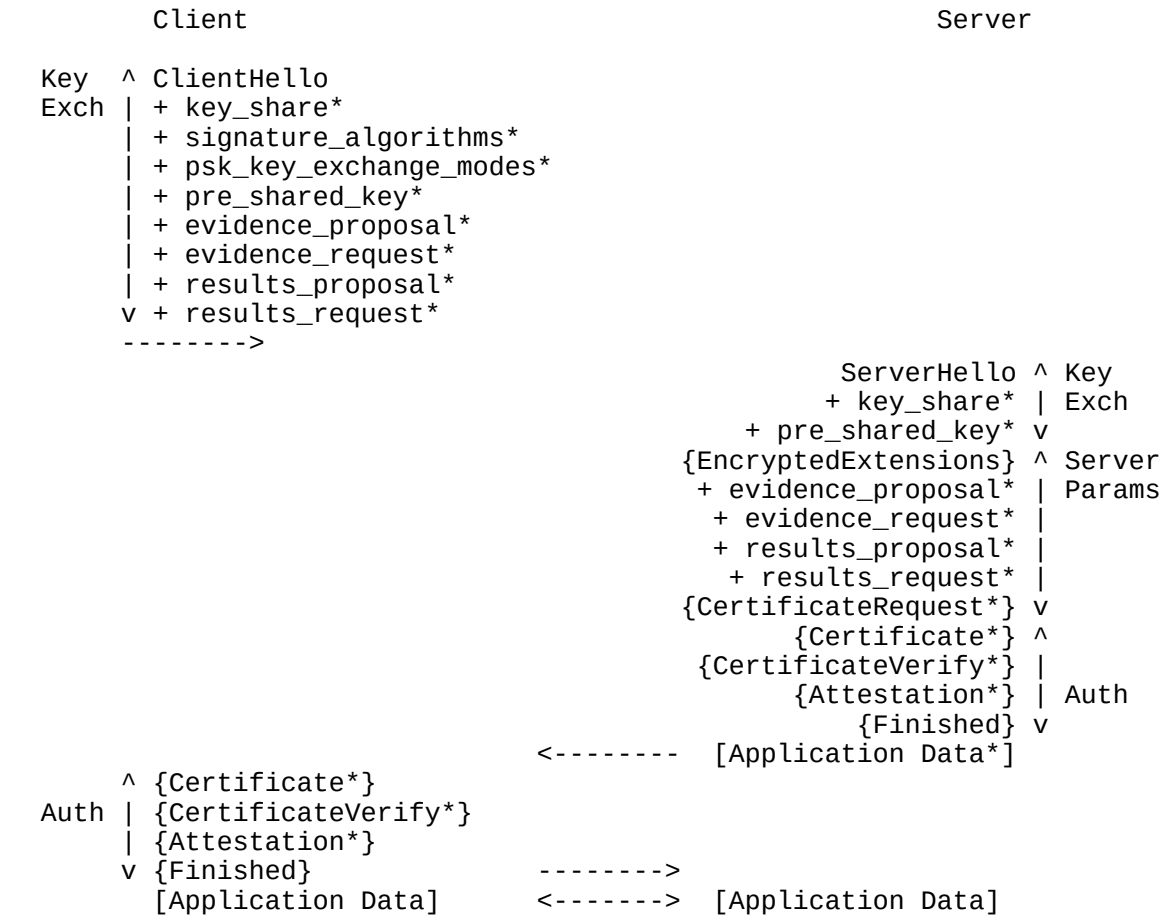

Figure 9: Attestation Message Overview.

9.1. Background Check Model

9.1.1. Client Hello

To indicate the support for passing Evidence in TLS following the Background Check Model, clients include the evidence_proposal and/or the evidence_request extensions in the ClientHello.

The evidence_proposal extension in the ClientHello message indicates the Evidence types the client is able to provide to the server.

The evidence_request extension in the ClientHello message indicates the Evidence types the client challenges the server to provide in an Attestation handshake message.

The evidence_proposal and evidence_request extensions sent in the ClientHello each carry a list of supported Evidence types, sorted by preference. When the client supports only one Evidence type, it is a list containing a single element.

The client MUST omit Evidence types from the evidence_proposal extension in the ClientHello if it cannot respond to a request from the server to present a proposed Evidence type, or if the client is not configured to use the proposed Evidence type with the given server. If the client has no Evidence types to send in the ClientHello it MUST omit the evidence_proposal extension in the ClientHello.

The client MUST omit Evidence types from the evidence_request extension in the ClientHello if it is not able to pass the indicated verification type to a Verifier. If the client does not act as a relying party with regards to Evidence processing (as defined in the RATS architecture) then the client MUST omit the evidence_request extension from the ClientHello.

9.1.2. Server Hello

If the server receives a ClientHello that contains the evidence_proposal extension and/or the evidence_request extension, then three outcomes are possible:

* The server does not support the extensions defined in this document. In this case, the server returns the EncryptedExtensions without the extensions defined in this document.

* The server supports the extensions defined in this document, but it does not have any Evidence type in common with the client. Then, the server terminates the session with a fatal alert of type "unsupported_evidence".

* The server supports the extensions defined in this document and has at least one Evidence type in common with the client. In this case, the processing rules described below are followed.

The evidence_proposal extension in the ClientHello indicates the Evidence types the client is able to provide to the server. If the server wants to request Evidence from the client, it MUST include the evidence_proposal extension in the EncryptedExtensions. This evidence_proposal extension in the EncryptedExtensions then indicates what Evidence format the client is requested to provide in an Attestation handshake message sent after the CertificateVerify message. The Evidence contained in the CMW payload MUST include a

binder derived from the TLS main secret and the message transcript up to ServerHello (see Section 5.6) in the TEE's signature, along with the client's TLS identity public key (TIK-C). The value conveyed in the evidence_proposal extension by the server MUST be selected from one of the values provided in the evidence_proposal extension sent in the ClientHello.

If none of the Evidence types supported by the client (as indicated in the evidence_proposal extension in the ClientHello) match the server-supported Evidence types, then the evidence_proposal extension in the ServerHello MUST be omitted.

The evidence_request extension in the ClientHello indicates what types of Evidence the client can challenge the server to return in an Attestation handshake message. With the evidence_request extension in the EncryptedExtensions, the server indicates the Evidence type carried in the Attestation handshake message sent after the CertificateVerify by the server. The Evidence contained in the CMW payload MUST include a binder derived from the TLS main secret and the message transcript up to ServerHello (see Section 5.6) in the TEE's signature, along with the server's TLS identity public key (TIK-S). The Evidence type in the evidence_request extension MUST contain a single value selected from the evidence_request extension in the ClientHello.

9.2. Passport Model

The results_proposal and results_request extensions are used to negotiate the protocol defined in this document, and in particular to negotiate the Verifier identities supported by each peer. These extensions are included in the ClientHello and ServerHello messages.

9.2.1. Client Hello

To indicate the support for passing Attestation Results in TLS following the Passport Model, clients include the results_proposal and/or the results_request extensions in the ClientHello message.

The results_proposal extension in the ClientHello message indicates the Verifier identities from which the client can relay Attestation Results. The client sends the Attestation Results in an Attestation handshake message after the CertificateVerify message.

The results_request extension in the ClientHello message indicates the Verifier identities from which the client expects the server to provide Attestation Results in an Attestation handshake message sent after the CertificateVerify.

The results_proposal and results_request extensions sent in the ClientHello each carry a list of supported Verifier identities, sorted by preference. When the client supports only one Verifier, it is a list containing a single element.

The client MUST omit Verifier identities from the results_proposal extension in the ClientHello if it cannot respond to a request from the server to present Attestation Results from a proposed Verifier, or if the client is not configured to relay the Results from the proposed Verifier with the given server. If the client has no Verifier identities to send in the ClientHello it MUST omit the results_proposal extension in the ClientHello.

The client MUST omit Verifier identities from the results_request extension in the ClientHello if it is not configured to trust Attestation Results issued by said verifiers. If the client does not act as a relying party with regards to the processing of Attestation Results (as defined in the RATS architecture) then the client MUST omit the results_request extension from the ClientHello.

9.2.2. Server Hello

If the server receives a ClientHello that contains the results_proposal extension and/or the results_request extension, then three outcomes are possible:

* The server does not support the extensions defined in this document. In this case, the server returns the EncryptedExtensions without the extensions defined in this document.

* The server supports the extensions defined in this document, but it does not have any trusted Verifiers in common with the client. Then, the server terminates the session with a fatal alert of type "unsupported_verifiers".

* The server supports the extensions defined in this document and has at least one trusted Verifier in common with the client. In this case, the processing rules described below are followed.

The results_proposal extension in the ClientHello indicates the Verifier identities from which the client is able to provide Attestation Results to the server. If the server wants to request Attestation Results from the client, it MUST include the results_proposal extension in the EncryptedExtensions. This results_proposal extension in the EncryptedExtensions then indicates what Verifier the client is requested to provide Attestation Results from in an Attestation handshake message sent after the

CertificateVerify message. The value conveyed in the results_proposal extension by the server MUST be selected from one of the values provided in the results_proposal extension sent in the ClientHello.

If none of the Verifier identities proposed by the client (as indicated in the results_proposal extension in the ClientHello) match the server-trusted Verifiers, then the results_proposal extension in the ServerHello MUST be omitted.

The results_request extension in the ClientHello indicates what Verifiers the client trusts as issuers of Attestation Results for the server. With the results_request extension in the EncryptedExtensions, the server indicates the identity of the Verifier who issued the Attestation Results carried in the Attestation handshake message sent after the CertificateVerify by the server. The Verifier identity in the results_request extension MUST contain a single value selected from the results_request extension in the ClientHello.

10. Security Considerations

TBD.

10.1. Security Guarantees

We note that as a pure cryptographic protocol, attested TLS as-is only guarantees that the Identity Key is known by the TEE. A number of additional guarantees must be provided by the platform and/or the TLS stack, and the overall security level depends on their existence and quality of assurance:

* The Identity Key is generated by the TEE.

* The Identity Key is never exported or leaked outside the TEE.

* The TLS protocol, whether implemented by the TEE or outside the TEE, is implemented correctly and (for example) does not leak any session key material.

These properties may be explicitly promised ("attested") by the platform, or they can be assured in other ways such as by providing source code, reproducible builds, formal verification etc. The exact mechanisms are out of scope of this document.

10.2. Freshness Guarantees

```
// TODO: Discuss freshness guarantees provided by secret derivation
// from the TLS main secret and message transcript.  Differences
// between Background Check and Passport mode.
```

10.3. Security of Reattestation After Extended Key Update

Reattestation relies on the assumption that both peers have derived the same Main Secret N+1 during the preceding EKU exchange. EKU by itself does not guarantee that the peers transitioned to a consistent key state in the presence of an active attacker. Deployments that require stronger guarantees will have use an authenticated transition mechanism discussed in [I-D.ietf-tls-extended-key-update] (e.g., post-handshake client authentication or Exported Authenticators) to detect key-schedule divergence before relying on reattestation results.

11. Privacy Considerations

In this section, we are assuming that the Attester is a TLS client, representing an individual person. We are concerned about the potential leakage of privacy sensitive information about that person, such as the correlation of different connections initiated by them.

In background-check mode, the Verifier not only has access to detailed information about the Attester’s TCB through Evidence, but it also knows the exact time and the party with whom the secure channel establishment is attempted (i.e., the RP). The privacy implications are similar to online OCSP [RFC6960]. While the RP may trust the Verifier not to disclose any information it receives, the same cannot be assumed for the Attester, which generally has no prior relationship with the Verifier. Some ways to address this include:

* Client-side redaction of privacy-sensitive evidence claims,
* Using selective disclosure (e.g., SD-JWT [I-D.ietf-oauth-selective-disclosure-jwt] with EAT [I-D.ietf-rats-eat]),
* Co-locating the Verifier role with the RP,
* Utilizing privacy-preserving attestation schemes (e.g., DAA [I-D.ietf-rats-daa]), or
* Utilizing Attesters manufactured with group identities (e.g., [FIDO-REQS]).

The latter two also have the property of hiding the peer’s identity from the RP.

Note that the equivalent of OCSP "stapling" involves using a passport topology where the Verifier’s involvement is unrelated to the TLS session.

Due to the inherent asymmetry of the TLS protocol, if the Attester acts as the TLS server, a malicious TLS client could potentially retrieve sensitive information from attestation Evidence without the client’s trustworthiness first being established by the server.

12. IANA Considerations

12.1. TLS Extensions

IANA is asked to allocate four new TLS extensions, evidence_request, evidence_proposal, results_request, results_proposal, from the "TLS ExtensionType Values" subregistry of the "Transport Layer Security (TLS) Extensions" registry [TLS-Ext-Registry]. These extensions are used in the ClientHello and the EncryptedExtensions messages. The values carried in these extensions are taken from TBD.

12.2. TLS Alerts

IANA is requested to allocate a value in the "TLS Alerts" subregistry of the "Transport Layer Security (TLS) Parameters" registry [TLS-Param-Registry] and populate it with the following entries:

* Value: TBD1
* Description: unsupported_evidence
* DTLS-OK: Y
* Reference: [This document]
* Comment:
* Value: TBD2
* Description: unsupported_verifiers
* DTLS-OK: Y
* Reference: [This document]
* Comment:

12.3. TLS Handshake Message Types

IANA is requested to allocate a new value in the "TLS HandshakeType" registry of the "Transport Layer Security (TLS) Parameters" registry [TLS-Param-Registry], as follows:

* Value: TBD
* Description: attestation
* DTLS-OK: Y
* Reference: [This document]
* Comment: Used to carry attestation Evidence or Attestation Results in the TLS handshake

13. Acknowledgements

We would like to thank Paul Howard, Arto Niemi, and Hannes Tschofenig for their contributions to earlier versions of this document.

14. References

14.1. Normative References

[I-D.ietf-rats-msg-wrap]
Birkholz, H., Smith, N., Fossati, T., Tschofenig, H., and D. Glaze, "RATS Conceptual Messages Wrapper (CMW)", Work in Progress, Internet-Draft, draft-ietf-rats-msg-wrap-23, 11 December 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-rats-msg-wrap-23>.

[I-D.ietf-tls-extended-key-update]
Tschofenig, H., T端xen, M., Reddy.K, T., Fries, S., and Y. Rosomakho, "Extended Key Update for Transport Layer Security (TLS) 1.3", Work in Progress, Internet-Draft, draft-ietf-tls-extended-key-update-07, 1 November 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-tls-extended-key-update-07>.

[I-D.ietf-tls-pake]
Bauman, L., Benjamin, D., Menon, S., and C. A. Wood, "A Password Authenticated Key Exchange Extension for TLS 1.3", Work in Progress, Internet-Draft, draft-ietf-tls-pake-00, 4 September 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-tls-pake-00>.

[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-14, 13 September 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-tls-rfc8446bis-14>.

[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>.

14.2. Informative References

[DICE-Layering]
Trusted Computing Group, "DICE Layering Architecture Version 1.00 Revision 0.19", July 2020, <https://trustedcomputinggroup.org/resource/dice-layering-architecture/>.

[FIDO-REQS]
Peirani, B. and J. Verrept, "FIDO Authenticator Security Requirements", November 2021, <https://fidoalliance.org/specs/fido-security-requirements/>.

[I-D.acme-device-attest]
Weeks, B., Mallaya, G., and S. Rajala, "Automated Certificate Management Environment (ACME) Device Attestation Extension", Work in Progress, Internet-Draft, draft-acme-device-attest-08, 7 December 2025, <https://datatracker.ietf.org/doc/html/draft-acme-device-attest-08>.

[I-D.fossati-tls-attestation]
Tschofenig, H., Sheffer, Y., Howard, P., Mihalcea, I., Deshpande, Y., Niemi, A., and T. Fossati, "Using Attestation in Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)", Work in Progress, Internet-Draft, draft-fossati-tls-attestation-09, 30 April 2025, <https://datatracker.ietf.org/doc/html/draft-fossati-tls-attestation-09>.

[I-D.ietf-oauth-selective-disclosure-jwt]
Fett, D., Yasuda, K., and B. Campbell, "Selective Disclosure for JWTs (SD-JWT)", Work in Progress, Internet-Draft, draft-ietf-oauth-selective-disclosure-jwt-22, 29 May 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-oauth-selective-disclosure-jwt-22>.

[I-D.ietf-rats-daa]
Birkholz, H., Newton, C., Chen, L., Giannetsos, T., and D. Thaler, "Direct Anonymous Attestation for the Remote Attestation Procedures Architecture", Work in Progress, Internet-Draft, draft-ietf-rats-daa-08, 3 September 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-rats-daa-08>.

[I-D.ietf-rats-eat]
Lundblade, L., Mandyam, G., O'Donoghue, J., and C. Wallace, "The Entity Attestation Token (EAT)", Work in Progress, Internet-Draft, draft-ietf-rats-eat-31, 6 September 2024, <https://datatracker.ietf.org/doc/html/draft-ietf-rats-eat-31>.

[I-D.ietf-teep-architecture]
Pei, M., Tschofenig, H., Thaler, D., and D. M. Wheeler, "Trusted Execution Environment Provisioning (TEEP) Architecture", Work in Progress, Internet-Draft, draft-ietf-teep-architecture-19, 24 October 2022, <https://datatracker.ietf.org/doc/html/draft-ietf-teep-architecture-19>.

[iana-content-formats]
IANA, "CoAP Content-Formats", <https://www.iana.org/assignments/core-parameters>.

[iana-media-types]
IANA, "Media Types", <https://www.iana.org/assignments/media-types>.

[RA-TLS] Knauth, T., Steiner, M., Chakrabarti, S., Lei, L., Xing, C., and M. Vij, "Integrating Remote Attestation with Transport Layer Security", January 2018, <https://arxiv.org/abs/1801.05863>.

[RFC6960] Santesson, S., Myers, M., Ankney, R., Malpani, A., Galperin, S., and C. Adams, "X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP", RFC 6960, DOI 10.17487/RFC6960, June 2013, <https://www.rfc-editor.org/rfc/rfc6960>.

[RFC9334] Birkholz, H., Thaler, D., Richardson, M., Smith, N., and W. Pan, "Remote ATtestation procedureS (RATS) Architecture", RFC 9334, DOI 10.17487/RFC9334, January 2023, <https://www.rfc-editor.org/rfc/rfc9334>.

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

[TLS-Param-Registry] IANA, "Transport Layer Security (TLS) Parameters", <https://www.iana.org/assignments/tls-parameters>.

[TPM1.2] Trusted Computing Group, "TPM Main Specification Level 2 Version 1.2, Revision 116", March 2011, <https://trustedcomputinggroup.org/resource/tpm-main-specification/>.

[TPM2.0] Trusted Computing Group, "Trusted Platform Module Library Specification, Family "2.0", Level 00, Revision 01.59", November 2019, <https://trustedcomputinggroup.org/resource/tpm-library-specification/>.

Appendix A. Document History

A.1. draft-fossati-seat-early-attestation-01

* Fix typo in key schedule. Clarify (again) that this is only adding to the schedule, not modifying any existing key derivations.

A.2. draft-fossati-seat-early-attestation-00

Initial version of draft-fossati-seat-early-attestation.

This version represents a major architectural change from [I-D.fossati-tls-attestation]. The key changes include:

* Removed certificate extension mechanism for conveying attestation Evidence

* Introduced new Attestation handshake message for carrying CMW (Conceptual Message Wrapper) payload

* Attestation message sent after CertificateVerify when server is attester

* Attestation message sent after CertificateVerify message when client is attester

* Removed use cases section

* Removed KAT (Key Attestation Token) and PAT (Platform Attestation Token) references, using CMW directly

* Nonces (client and server) and attester’s TLS identity public key are included in TEE-signed Evidence/AttestationResults within CMW

* CertificateVerify remains unchanged from baseline TLS (no proof-of-possession needed)

* Added session resumption discussion (resumption MUST be rejected if reattestation is required per local policy)

* Added reattestation

Appendix B. Design Rationale

This appendix explains the rationale for introducing a dedicated Attestation handshake message, instead of embedding attestation in an extension inside the TLS Certificate message. That approach fails to meet key security, and privacy requirements.

B.1. Requires Certificate Authentication

TLS 1.3 supports authentication modes where no Certificate message is sent:

* PSK-based authentication

* PAKE-based authentication [I-D.ietf-tls-pake]

A design that relies on a Certificate message extension cannot operate in these cases. In contrast, a dedicated Attestation handshake message works regardless of authentication mode, making it compatible with the full TLS authentication spectrum.

B.2. Reattestation Not Fully Supported

TLS allows Post-Handshake client authentication Section 4.2.6 of [I-D.ietf-tls-rfc8446bis] but provides no mechanism for Post-Handshake server authentication. As a result, a design that embeds attestation inside the Certificate message would allow only the client and not the server to refresh its attestation. This is insufficient for deployments that require periodic server

reattestation.

Authors' Addresses

Yaron Sheffer
Intuit
Email: yaronf.ietf@gmail.com

Ionut Mihalcea
Arm Limited
Email: Ionut.Mihalcea@arm.com

Yogesh Deshpande
Arm Limited
Email: Yogesh.Deshpande@arm.com

Thomas Fossati
Linaro
Email: thomas.fossati@linaro.org

Tirumaleswar Reddy
Nokia
Email: k.tirumaleswar_reddy@nokia.com