TLS
Internet-Draft
Intended status: Standards Track
Expires: 28 November 2026

Y. Sheffer
Intuit
I. Mihalcea
Y. Deshpande
Arm Limited
T. Fossati
Linaro
T. Reddy
Nokia
27 May 2026

Using Attestation in Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)
draft-fossati-seat-early-attestation-04

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 TLS extensions 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 extensions to the TLS handshake that enable the binding of the TLS connection and its 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 attestation extension.

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 6.1 and Section 6.2.

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:

The terms "appraise" and "verify" are used with distinctive semantics throughout the document:

"Appraise" covers the act of checking the validity of Attestation Results or Evidence, as per [RFC9334], performed by Relying Parties and Verifiers respectively. "Verify" covers all other checks performed by the two TLS peers, intended to assess the correctness of the cryptographic and protocol operations of the TLS layer.

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 nonce value provided by the TLS stack to the TEE. It is used for binding attestation Evidence to a specific TLS handshake and for providing freshness.

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. A new Attestation extension is introduced to the Certificate message. This extension carries attestation Evidence or Attestation Results.

* Independent key derivation: Binder derivation for attestation (see Section 5.1) is completely independent of the regular TLS key schedule. 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 an Attestation Binder mechanism (see Section 5.1) 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 extension (see Section 4.1). The CMW payload in the Attestation extension 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 the Verifier’s signature over AttestationResults within the CMW MUST include an attestation binder derived from the message transcript (see Section 5.1) 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 extension (in the Passport Model), or by relaying the Evidence from the Attestation extension to the Verifier and receiving the Attestation Results. Subsequent verification of possession of the attested key in the CertificateVerify message remains unchanged from baseline TLS.

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

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 8.1 for more details.

4.1. Attestation Extension

As defined in Section 4.4.2 of [I-D.ietf-tls-rfc8446bis], the TLS Certificate message contains a certificate_list, which is a sequence of CertificateEntry structures.

When attestation is negotiated via the extensions defined in this document, the attestation extension defined in this document MUST appear only in the first CertificateEntry of the Certificate message and applies exclusively to the end-entity certificate.

The extension MUST NOT appear in any other CertificateEntry.

If the attestation extension is received in any other position, the receiver MUST abort the handshake with a fatal illegal_parameter alert.

This message carries a CMW (Conceptual Message Wrapper) payload as defined in [I-D.ietf-rats-msg-wrap].

The attestation extension structure is defined as follows:

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

Figure 1: Attestation Extension 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 ensuring that the attestation is associated with this particular TLS connection, as well as 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 derivation from both peers' randomness. See Section 5.1 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.

As noted, each peer's attestation is carried in the Attestation extension within that peer's Certificate message. This section describes how the attestation is produced, bound to the TLS handshake and verified by the recipient.

5.1. Cryptographic Operations

The cryptographic operations defined in this section bind attestation Evidence to a specific TLS handshake. This binding prevents replay and relay of attestation Evidence across different TLS connections, and ensures that attestation Evidence presented during a handshake corresponds to the authenticated TLS session in which it is conveyed.

The attestation Evidence or Attestation Results are generated by a TEE and signed using an attestation key. The signed Evidence includes inputs originating from different trust domains.

The attestation binder is provided by the TLS stack and serves as a nonce that ensures freshness and binding to a specific TLS handshake, as well as binding to the attester's TLS public key.

5.1.1. Attestation Binder Definition

The attestation binder is computed using primitives defined in Section 4.4.1 and 7.1 of [I-D.ietf-tls-rfc8446bis].

Both peers derive a single attestation base from the same transcript checkpoint, ClientHello...ServerHello.

```
attest_base = HKDF-Expand-Label(0, "attestation base",
                                Hash(ClientHello...ServerHello), Hash.length)

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

TLS_Client_Public_Key and TLS_Server_Public_Key denote the DER-encoded SubjectPublicKeyInfo of the peer's end-entity certificate. Hash is the cipher suite hash function for the handshake (Section 7.1 of [I-D.ietf-tls-rfc8446bis]).

We note that HKDF-Expand-Label is used to produce binding values rather than keying material. HKDF-Extract is not invoked, as there is no input key material to combine. The "0" parameter denotes a byte string of Hash.length zeroes.

5.1.2. Verification

Upon receipt of an attestation extension, the peer MUST compute the attestation binder.

If the peer's Evidence is rejected (binder mismatch, failed Evidence appraisal, or malformed CMW), the receiver MUST send an attestation_failed fatal alert and abort the handshake (see Section 10.2).

Depending on the architecture (see also Section 5.3), either the peer verifies the binding or else it delegates this responsibility to an external Verifier.

* In the former case, the peer MUST compare the computed binder value to the attestation binder included in the signed Evidence or signed Attestation Results. If the values do not match, the peer MUST treat the attestation as invalid.

* In the latter case, the RP MUST convey the binder to the Verifier. The Verifier MUST appraise that the conveyed binder is identical to the one that was signed in the Evidence or Attestation Results. If appraisal fails, the receiver MUST treat the attestation as invalid.

// TODO: define a way to transport the binder to a remote Verifier.
// Possibly as a (new) conceptual message (CM) within a collection.
// This would provide the Verifier whatever information it cannot
// compute on its own, while not forcing the TLS stack to parse the
// Evidence.

5.1.3. Security Properties

Binding attestation Evidence to the TLS handshake transcript hash provides the following security properties:

* Replay protection: Evidence generated for a previous handshake cannot be reused in a later handshake.

* Relay protection: Evidence obtained from one TLS connection cannot be successfully presented in a different TLS connection, even in the presence of a MiTM attacker.

In typical deployments where the TLS handshake executes outside the TEE, a compromised host can execute the TLS handshake in the rich operating system and use the TEE as a signing oracle by presenting the attestation binder value to obtain valid-looking attestation Evidence.

However an endorsed TEE (one that is operating as required by this protocol) is required to verify the binder against the TLS public key associated with the private key that it holds. This verification, in conjunction with the TEE’s endorsement being appraised, ensures that relay attacks are prevented.

The attestation binder prevents replay of Evidence across TLS connections. The binding to the TLS identity key ensures that Evidence produced by one endpoint cannot be replayed in a TLS connection involving a different endpoint, as the verifier checks that the binder matches the public key presented in the current TLS connection. The additional binding to the transcript through ClientHello and ServerHello ensures that Evidence cannot be replayed across TLS connections, as ClientHello.random and ServerHello.random are independently generated by each peer for every TLS connection.

5.2. 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 signed Evidence MUST include an Attestation Binder generated using the hash of the TIK public key (TIK_pub_hash) (see Section 5.1).

The Relying Party MUST compute the hash of the TIK public key extracted from the TLS end-entity certificate using the same hash algorithm and verify that it matches the TIK_pub_hash included in the Evidence. Successful verification binds the attestation Evidence to the TLS identity used for authentication. This verification is performed by the Relying Party, as the Verifier may not be co-located with the Relying Party and may not have access to the TLS handshake or the TLS end-entity certificate, consistent with the RATS architecture. Alternatively, in deployments where the Verifier is not co-located with the Relying Party, the Relying Party MAY supply the Verifier with the hash of the TIK public key. The Verifier then compares this value with the TIK public key hash included in the Evidence. If the values do not match, the attestation MUST be considered invalid.

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 - in and of itself - ensure security of the TLS stack when the stack is outside the TEE, and see Section 8.1 for a further discussion.

5.3. 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.

Architecturally we propose to add a thin shim between the traditional TLS stack and the TEE as shown in Figure 2. Implementations will choose whether to incorporate the shim into the TEE (making for a "smarter" TEE and better protection for the remote attestation protocol), or in case of a legacy TEE that cannot be modified, the shim can be added to the TLS stack.

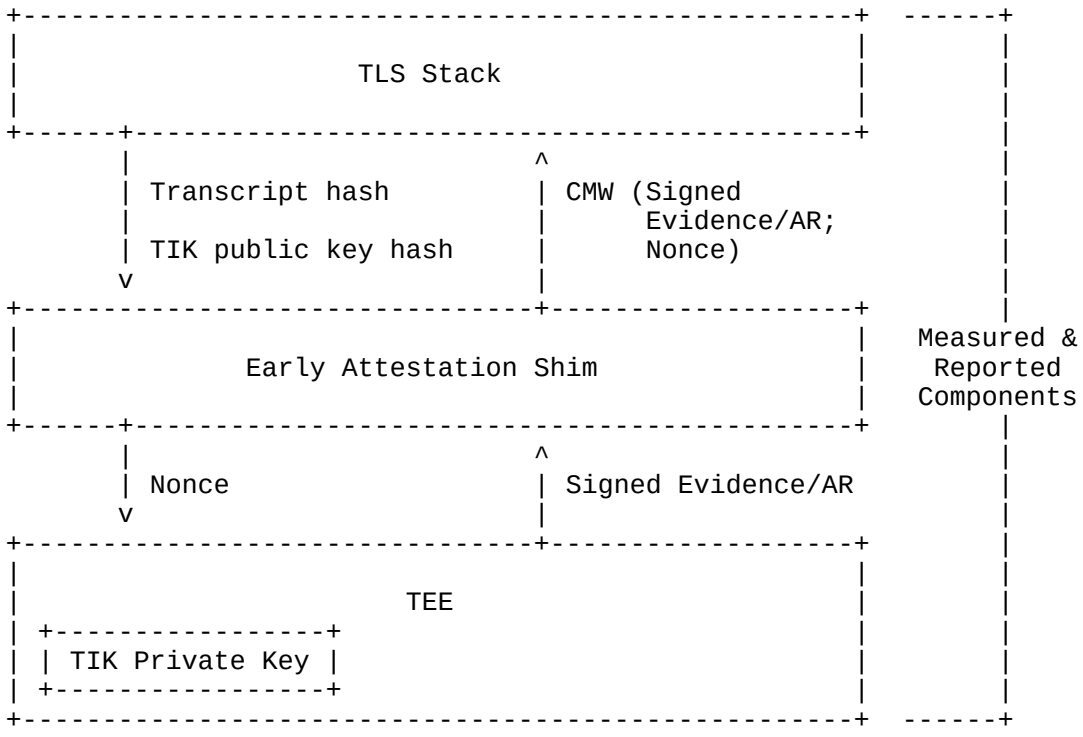

Figure 2: TLS Stack Interface with the TEE

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 5.1, 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.

* As shown in the diagram, the TEE itself as well as the TLS stack and the shim SHOULD all be measured and reported as part of the platform's remote attestation.

5.4. Reattestation

Attestation Evidence or Attestation Results may become stale over time. For long-lived TLS connections, a relying party may require updated assurance that the peer continues to operate in a trustworthy state.

5.4.1. Post-Handshake Reattestation Using Client Authentication

Post-handshake client authentication defined in Section 4.6.2 of [I-D.ietf-tls-rfc8446bis] can be used to obtain updated attestation Evidence or Attestation Results from the TLS client. In this case, the TLS server sends a CertificateRequest message after the TLS handshake authentication. The client responds with the standard TLS authentication messages (Certificate, CertificateVerify, and Finished). If attestation has been negotiated for the TLS connection, the client includes the attestation extension in the Certificate message carrying updated Evidence or Attestation Results.

The attestation binder can be derived from the post-handshake authentication transcript defined in Section 4.4 of [I-D.ietf-tls-rfc8446bis].

This mechanism allows a server to request updated attestation from the client. However, TLS currently does not define a mechanism for post-handshake server authentication. To address this limitation, the subsequent sections discuss design options for handling attestation freshness.

5.4.2. Option 1: Carrying Attestation in Extended Key Update

One possible approach is to extend the Extended Key Update (EKU) mechanism by introducing a new ExtendedKeyUpdate message subtype to carry attestation Evidence or Attestation Results.

However, this approach tightly couples attestation to EKU, even though the two serve different purposes.

5.4.3. Option 2: No Reattestation (Reconnect for Freshness)

Another approach is to not support reattestation within an established TLS connection. When fresh attestation is required, the client and server terminate the existing TLS session and establish a new one, during which fresh Evidence or Attestation Results are exchanged as part of the handshake.

This approach keeps the TLS protocol unchanged and avoids introducing post-handshake mechanisms. However, it will be disruptive for long-lived TLS connections.

5.4.4. Option 3: Post-Handshake Reattestation Using CertificateUpdate

In this design, reattestation is supported using the CertificateUpdate message defined in [I-D.rosomakho-tls-cert-update]. Under this approach, the attester sends a CertificateUpdate message carrying a new Certificate message with updated attestation information. The refreshed attestation is bound to the existing TLS session using post-handshake TLS context.

6. 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 appraised during the handshake, and the Passport Model, where pre-appraised 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.
```

6.1. Evidence Extensions (Background Check Model)

The EvidenceType structure contains an indicator for the type of Evidence expected in the Attestation extension. The Evidence contained in the CMW payload is sent in the Attestation extension (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 3: 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].

6.2. 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 4: TLS Extension Structure for Attestation Results.

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

7. TLS Client and Server Handshake Behavior

The high-level message exchange in Figure 5 shows the evidence_proposal, evidence_request, results_proposal, and results_request extensions added to the ClientHello and the EncryptedExtensions messages.

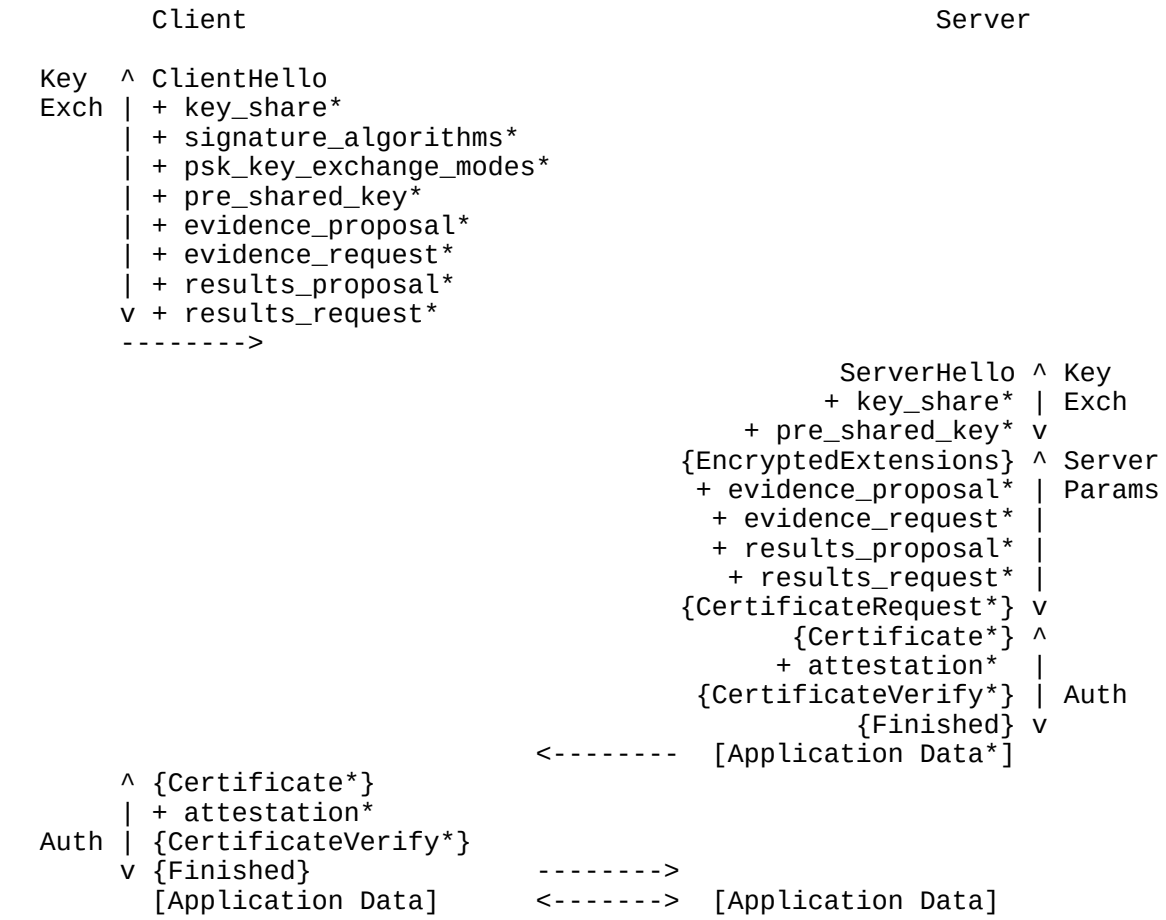

Figure 5: Early Attestation Handshake Overview

7.1. Background Check Model

7.1.1. 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 extension.

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.

7.1.2. 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 extension in the Certificate message. The signed Evidence contained in the CMW payload MUST include an Attestation

Binder as a nonce value (see Section 5.1) in the TEE's signature. 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 extension. With the evidence_request extension in the EncryptedExtensions, the server indicates the Evidence type carried in the Attestation extension sent after the CertificateVerify by the server. The signed Evidence contained in the CMW payload MUST include an Attestation Binder as a nonce value (see Section 5.1) in the TEE's signature. The Evidence type in the evidence_request extension MUST contain a single value selected from the evidence_request extension in the ClientHello.

7.2. 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.

7.2.1. 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 extension in the Certificate 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 extension in the Certificate message.

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.

7.2.2. 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 extension in the Certificate 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 extension sent in the Certificate 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.

8. Security Considerations

TBD.

8.1. 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.

8.2. Freshness Guarantees

// TODO: Discuss freshness guarantees provided by the Attestation
// Binder. Differences between Background Check and Passport mode.

9. 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.

10. IANA Considerations

10.1. TLS Extensions

IANA is asked to allocate five new TLS extensions, attestation, 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.

10.2. TLS Alerts

IANA is requested to allocate values 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:
* Value: TBD3
* Description: attestation_failed
* DTLS-OK: Y
* Reference: [This document]
* Comment:

11. Acknowledgements

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

12. References

12.1. 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-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>.

12.2. 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-09, 2 March 2026, <https://datatracker.ietf.org/doc/html/draft-ietf-rats-daa-09>.

[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>.

[I-D.rosomakho-tls-cert-update]
Rosomakho, Y. and T. Reddy.K, "Certificate Update in TLS 1.3", Work in Progress, Internet-Draft, draft-rosomakho-tls-cert-update-01, 21 December 2025, <https://datatracker.ietf.org/doc/html/draft-rosomakho-tls-cert-update-01>.

[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-04

* Register the attestation_failed alert for Evidence verification failure after the attestation extension is processed; clarify roles of the three attestation-related alerts in Section 10.2.

* Hash TLS public keys in HKDF-Expand-Label context so HkdfLabel stays within the 255-octet limit (post-quantum public keys); see Section 5.1.

* Simplify attestation binder derivation to a single shared transcript checkpoint (ClientHello...ServerHello) for both peers (see Section 5.1).

* Replaced Derive-Secret with HKDF-Expand-Label

A.2. draft-fossati-seat-early-attestation-03

* Replace the Attestation message by an Attestation (certificate) extension, to bring this protocol within the requirements of the SEAT charter.

* Define the attestation binder and decouple it from the TLS key schedule.

* List multiple design options for reattestation.

* Add architecture diagram for TLS stack interface with the TEE.

* Add defense-in-depth guidance for measuring TEE, TLS stack, and shim.

* Remove various outdated sections.

A.3. draft-fossati-seat-early-attestation-02

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

A.4. draft-fossati-seat-early-attestation-01

(Submitted by mistake.)

A.5. 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

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