

Workgroup: TLS

Internet-Draft:

draft-fossati-tls-attestation-02

Published: 24 October 2022

Intended Status: Standards Track

Expires: 27 April 2023

Authors: H. Tschofenig    T. Fossati    P. Howard  
          Arm Limited        Arm Limited    Arm Limited  
          I. Mihalcea      Y. Deshpande  
          Arm Limited      Arm Limited

## **Using Attestation in Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)**

### **Abstract**

Attestation is the process by which an entity produces evidence about itself that another party can use to evaluate the trustworthiness of that entity.

In use cases that require the use of remote attestation, such as confidential computing or device onboarding, an attester has to convey evidence or attestation results to a relying party. This information exchange may happen at different layers in the protocol stack.

This specification provides a generic way of passing evidence and attestation results in the TLS handshake. Functionality-wise this is accomplished with the help of key attestation.

### **Status of This Memo**

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 27 April 2023.

## Copyright Notice

Copyright (c) 2022 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Revised BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.

## Table of Contents

- [1. Introduction](#)
- [2. Conventions and Terminology](#)
- [3. Overview](#)
- [4. Use of Evidence with the Background Check Model](#)
  - [4.1. TLS Client as Attester](#)
  - [4.2. TLS Server as Attester](#)
- [5. Evidence Extension \(Background Check Model\)](#)
- [6. TLS Client and Server Handshake Behavior](#)
  - [6.1. Client Hello](#)
  - [6.2. Server Hello](#)
- [7. Background-Check Model Examples](#)
  - [7.1. Cloud Confidential Computing](#)
  - [7.2. IoT Device Onboarding](#)
- [8. Security Considerations](#)
- [9. IANA Considerations](#)
  - [9.1. TLS Extensions](#)
  - [9.2. TLS Alerts](#)
  - [9.3. TLS Certificate Types](#)
- [10. References](#)
  - [10.1. Normative References](#)
  - [10.2. Informative References](#)
- [Appendix A. History](#)
  - [A.1. draft-fossati-tls-attestation-02](#)
  - [A.2. draft-fossati-tls-attestation-01](#)
  - [A.3. draft-fossati-tls-attestation-00](#)
- [Appendix B. Working Group Information](#)
- [Authors' Addresses](#)

## 1. Introduction

The Remote ATtestation Procedures (RATS) architecture defines two basic types of topological patterns to communicate between an

attester, a relying party, and a verifier, namely the background-check model and the passport model. These two models are fundamentally different and require a different treatment when incorporated into the TLS handshake. For better readability to use different extensions for these two models.

The two models can be summarized as follows:

\*In the background check model, the attester conveys evidence to the relying party, which then forwards the evidence to the verifier for appraisal; the verifier computes the attestation result and sends it back to the relying party.

\*In the passport model, the attester transmits evidence to the verifier directly and receives attestation results, which are then relayed to the relying party. This specification supports both patterns.

Several formats for encoding evidence are available, such as - the Entity Attestation Token (EAT) [[I-D.ietf-rats-eat](#)], - the Trusted Platform Modules (TPMs) [[TPM1.2](#)] [[TPM2.0](#)], - the Android Key Attestation, and - Apple Key Attestation.

Like-wise, there are different encodings available for attestation results. One such encoding, AR4SI [[I-D.ietf-rats-ar4si](#)] is being standardized by the RATS working group.

This version of the specification defines how to support the background check model in the TLS handshake, such that the details about the attestation technology are agnostic to the TLS handshake itself. Later versions of the specification will support the passport model as well.

To give the peer information that the handshake signing key is properly secured, the associated evidence has to be verified by that peer. Hence, attestation evidence about the security state of the signing key is needed, which is typically associated with evidence about the overall platform state. The platform attestation service ensures that the key attestation service has not been tampered with. The platform attestation service issues the Platform Attestation Token (PAT) and the key attestation service issues the Key Attestation Token (KAT). The security of the protocol critically depends on the verifiable binding between these two logically separate units of evidence.

This document does not define how different attestation technologies are encoded. This has either already been done or is being accomplished by companion specifications.

## 2. Conventions and Terminology

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

## 3. Overview

The Remote Attestation Procedures (RATS) architecture [[I-D.ietf-rats-architecture](#)] defines two types of interaction models for attestation, namely the passport model and the background check model. The subsections below explain the difference in their interactions.

As typical with new features in TLS, the client indicates support for the new extension in the ClientHello. The newly introduced extensions allow evidence and nonces to be exchanged. The nonces are used for guaranteeing freshness of the exchanged evidence.

When the evidence extension is successfully negotiated, the content of the Certificate message contains a payload that is encoded based on the wrapper defined in [[I-D.ftbs-rats-msg-wrap](#)].

In TLS a client has to demonstrate possession of the private key via the CertificateVerify message, when client-based authentication is requested. The attestation payload must contain a key attestation token, which associates a private key with the attestation information. An example of a key attestation token format utilizing the EAT-format can be found in [[I-D.bft-rats-kat](#)].

The recipient extracts evidence from the Certificate message and relays it to the verifier to obtain attestation results. Subsequently, the attested key is used to verify the CertificateVerify message.

## 4. Use of Evidence with the Background Check Model

The background check model is described in Section 5.2 of [[I-D.ietf-rats-architecture](#)] and allows the following modes of operation when used with TLS, namely

- \*TLS client is the attester,
- \*TLS server is the attester, and
- \*TLS client and server mutually attest each other.

We will show the message exchanges of the three cases in subsections below.

#### 4.1. TLS Client as Attester

In this use case the TLS client, as the attester, is challenged by the TLS server to provide evidence. The TLS client is the attester and the the TLS server acts as a relying party. The TLS server needs to provide a nonce in the EncryptedExtensions to the TLS client so that the attestation service can feed the nonce into the generation of the evidence. 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.

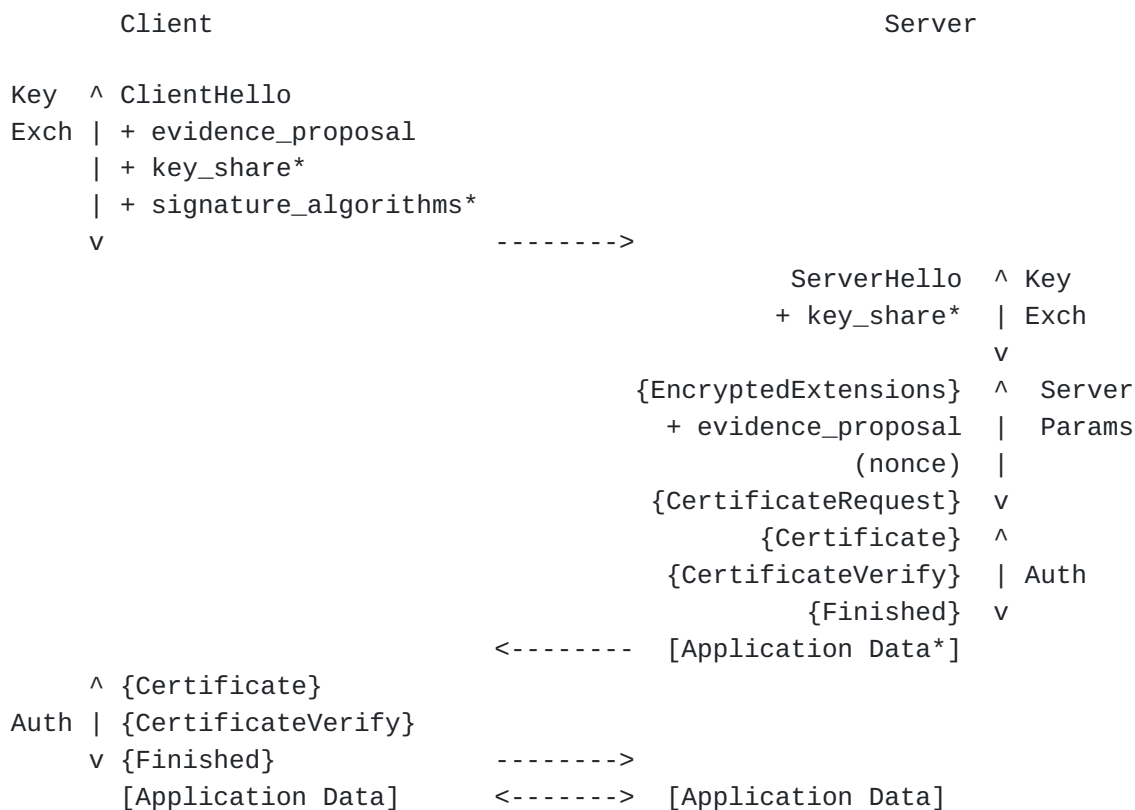


Figure 1: TLS Client Providing Evidence to TLS Server.

#### 4.2. TLS Server as Attester

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 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 ensured that the confidential computing platform is genuine.



Figure 2: TLS Client Providing Evidence to TLS Server.

## 5. Evidence Extension (Background Check Model)

This document defines two new extensions, the `evidence_request` and the `evidence_proposal`, for use with the background check model.

The `EvidenceType` structure encodes either a media type or as a content format. The media type is a string-based identifier while the content format uses a number. The former is more flexible and does not necessarily require a registration through IANA while the latter is more efficient over-the-wire.

```

enum { NUMERIC(0), STRING(1) } encodingType;

struct {
    encodingType type;
    select (encodingType) {
        case NUMERIC:
            uint16 content_format;
        case STRING:
            opaque media_type<0..2^16-1>;
    };
} EvidenceType;

struct {
    select(ClientOrServerExtension) {
        case client:
            EvidenceType supported_evidence_types<1..2^8-1>;
            opaque nonce<0..2^8-1>;

        case server:
            EvidenceType selected_evidence_type;
    }
} evidenceRequestTypeExtension;

struct {
    select(ClientOrServerExtension) {
        case client:
            EvidenceType supported_evidence_types<1..2^8-1>;

        case server:
            EvidenceType selected_evidence_type;
            opaque nonce<0..2^8-1>;
    }
} evidenceProposalTypeExtension;

```

Figure 3: TLS Structure for Evidence.

The Certificate payload is used as a container, as shown in [Figure 4](#), and follows the model of [[RFC8446](#)].

```

struct {
    select (certificate_type) {
        case RawPublicKey:
            /* From RFC 7250 ASN.1_subjectPublicKeyInfo */
            opaque ASN1_subjectPublicKeyInfo<1..2^24-1>;

            /* payload used to convey evidence */
        case attestation:
            opaque evidence<1..2^24-1>;

        case X509:
            opaque cert_data<1..2^24-1>;
    };
    Extension extensions<0..2^16-1>;
} CertificateEntry;

struct {
    opaque certificate_request_context<0..2^8-1>;
    CertificateEntry certificate_list<0..2^24-1>;
} Certificate;

```

Figure 4: Certificate Message.

The encoding of the evidence structure is defined in [\[I-D.ftbs-rats-msg-wrap\]](#).

## 6. TLS Client and Server Handshake Behavior

The high-level message exchange in [Figure 5](#) shows the evidence\_proposal and evidence\_request extensions added to the ClientHello and the EncryptedExtensions messages.





Figure 5: Attestation Message Overview.

### 6.1. 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` indicates the evidence types the client is able to provide to the server, when requested using a `CertificateRequest` message.

The `evidence_request` extension in the `ClientHello` indicates the types of evidence types the client challenges the server to provide in a subsequent `Certificate` payload.

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.

## 6.2. 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 extension defined in this document. In this case, the server returns the EncryptedExtensions without the extensions defined in this document.
- \*The server supports the extension 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, when challenged using a certificate\_request message. If the server wants to request evidence from the client, it MUST include the client\_attestation\_type extension in the EncryptedExtensions. This evidence\_proposal extension in the EncryptedExtensions then indicates what evidence format the client is requested to provide in a subsequent Certificate message. 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. The server MUST also send a certificate\_request message.

If the server does not send a certificate\_request message or 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 a subsequent `Certificate` message. With the `evidence_request` extension in the `EncryptedExtensions`, the server indicates the evidence type carried in the `Certificate` message sent by the server. The evidence type in the `evidence_request` extension MUST contain a single value selected from the `evidence_request` extension in the `ClientHello`.

## **7. Background-Check Model Examples**

### **7.1. Cloud Confidential Computing**

In this example, a confidential workload is executed on computational resources hosted at a cloud service provider. This is a typical scenario for secure, privacy-preserving multiparty computation, including anti-money laundering, drug development in healthcare, contact tracing in pandemic times, etc.

In such scenarios, the users (e.g., the party providing the data input for the computation, the consumer of the computed results, the party providing a proprietary ML model used in the computation) have two goals:

- \*Identifying the workload they are interacting with,

- \*Making sure that the platform on which the workload executes is a Trusted Execution Environment (TEE) with the expected features.

A convenient arrangement is to verify that the two requirements are met at the same time that the secure channel is established.

The protocol flow, alongside all the involved actors, is captured in [Figure 6](#) where the TLS client is the user (the relying party) while the TLS server is co-located with the TEE-hosted confidential workload (the attester).

The flow starts with the client initiating a verification session with a trusted verifier. The verifier returns the kinds of evidence it understands and a nonce that will be used to challenge the attester.

The client starts the TLS handshake with the server by supplying the attestation-related parameters it has obtained from the verifier. If the server supports one of the offered evidence types, it will echo it in the `specular` extension and proceed by invoking the local API to request the attestation. The returned evidence binds the identity key with the platform identity and security state. The server then signs the handshake transcript with the (attested) identity key, and

sends the attestation evidence together with the signature over to the client.

The client forwards the attestation evidence to the verifier using the previously established session, obtains the attestation result and checks it is acceptable according to its local policy. If so, it proceeds and verifies the handshake signature using the corresponding public key (for example, using the PoP key in the KAT evidence [[I-D.bft-rats-kat](#)]).

The attestation evidence verification combined with the verification of the CertificateVerify signature provide confirmation that the presented cryptographic identity is bound to the workload and platform identity, and that the workload and platform are trustworthy. Therefore, after the handshake is finalized, the client can trust the workload on the other side of the established secure channel to provide the required confidential computing properties.

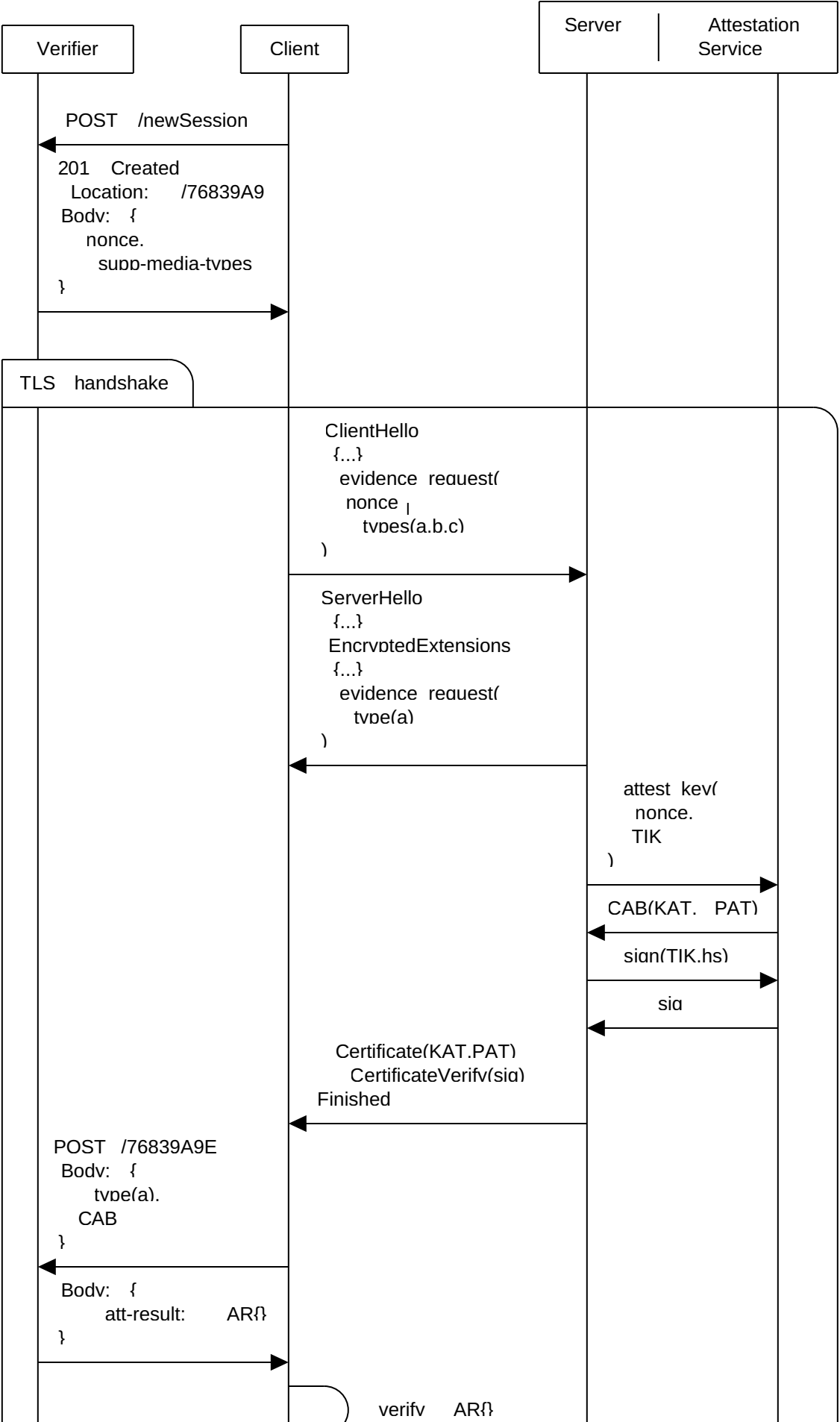


Figure 6: Example Exchange with Server as Attester.

## 7.2. IoT Device Onboarding

In this example, an IoT is onboarded to a cloud service provider (or to a network operator). In this scenario there is typically no a-priori relationship between the device and the cloud service provider that will remote manage the device.

In such scenario, the cloud service provider wants to make sure that the device runs the correct version of firmware, has not been rooted, is emulated, or cloned.

The protocol flow is shown in [Figure 7](#) where the client is the attester while the server is the relying party.

The flow starts with the client initiating a TLS exchange with the TLS server operated by the cloud service provider. The client indicates what evidence types it supports.

The server obtains a nonce from the verifier, in real-time or from a reserved nonce range, and returns it to the client alongside the selected evidence type. Since the evidence will be returned in the Certificate message the server has to request mutual authentication via the CertificateRequest message.

The client, when receiving the EncryptedExtension with the evidence\_proposal, will proceed by invoking a local API to request the attestation. The returned evidence binds the identity key with the workload and platform identity and security state. The client then signs the handshake transcript with the (attested) identity key, and sends the evidence together with the signature over to the server.

The server forwards the attestation evidence to the verifier, obtains the attestation result and checks it is acceptable according to its local policy. The evidence verification combined with the verification of the CertificateVerify signature provide confirmation that the presented cryptographic identity is bound to the platform identity, and that the platform is trustworthy.

If successful, the server proceeds with the application layer protocol exchange. If, for some reason, the attestation result is not satisfactory the TLS server will terminate the exchange.

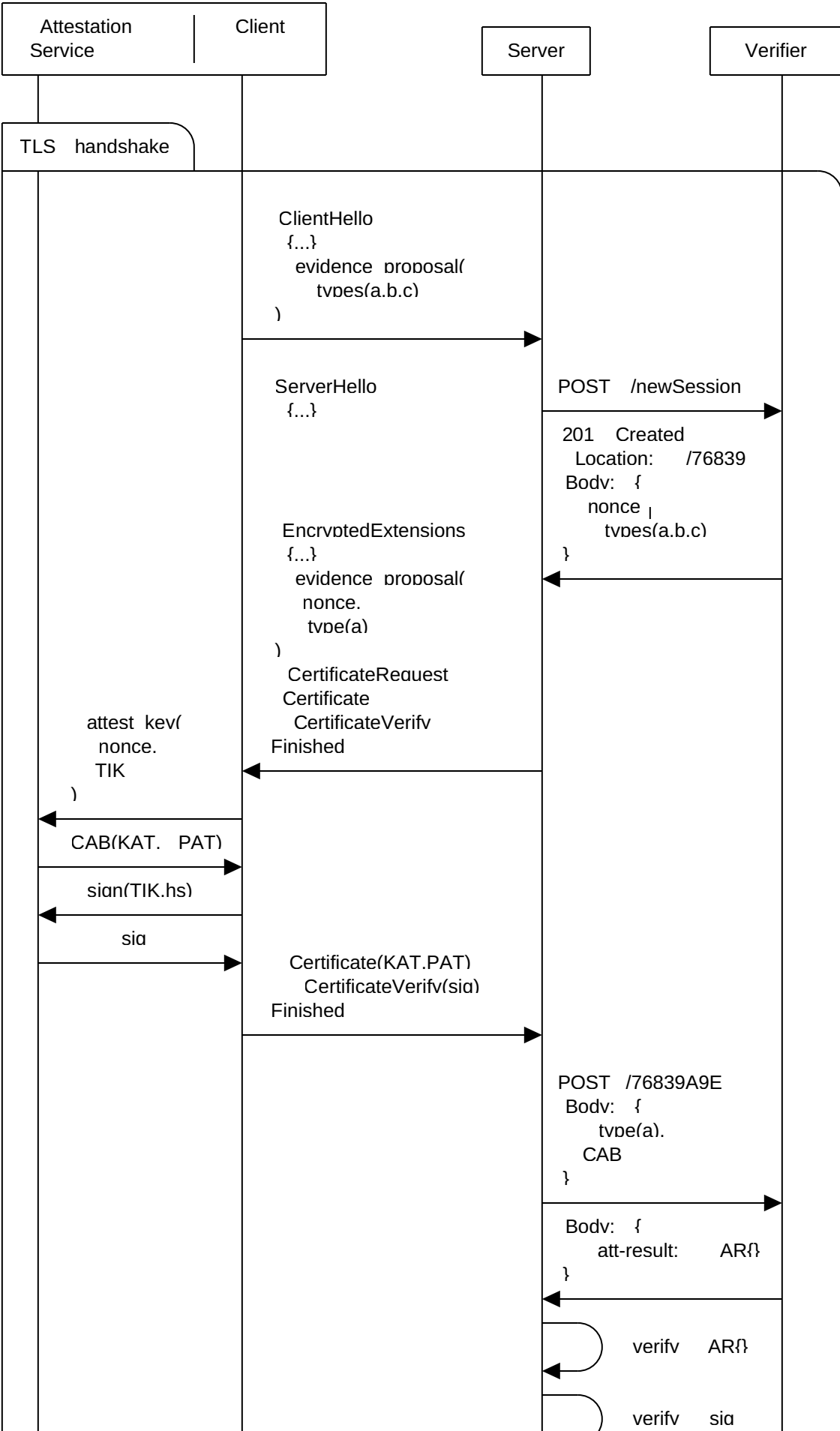


Figure 7: Example Exchange with Client as Attester.

## 8. Security Considerations

TBD.

## 9. IANA Considerations

### 9.1. TLS Extensions

IANA is asked to allocate two new TLS extensions, `evidence_request` and `evidence_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.

### 9.2. 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 entry:

\*Value: TBD

\*Description: `unsupported_evidence`

\*DTLS-OK: Y

\*Reference: [This document]

\*Comment:

### 9.3. TLS Certificate Types

IANA is requested to allocate a new value in the "TLS Certificate Types" subregistry of the "Transport Layer Security (TLS) Extensions" registry [[TLS-Ext-Registry](#)], as follows:

\*Value: TBD2

\*Description: `Attestation`

\*Reference: [This document]

## 10. References

### 10.1. Normative References

[I-D.bft-rats-kat]



Brossard, M., Fossati, T., and H. Tschofenig, "An EAT-based Key Attestation Token", Work in Progress, Internet-Draft, draft-bft-rats-kat-00, 21 October 2022, <<https://www.ietf.org/archive/id/draft-bft-rats-kat-00.txt>>.

[I-D.ftbs-rats-msg-wrap] Birkholz, H., Smith, N., Fossati, T., and H. Tschofenig, "RATS Conceptual Messages Wrapper", Work in Progress, Internet-Draft, draft-ftbs-rats-msg-wrap-01, 20 October 2022, <<https://www.ietf.org/archive/id/draft-ftbs-rats-msg-wrap-01.txt>>.

[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/info/rfc2119>>.

[RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <<https://www.rfc-editor.org/info/rfc8446>>.

## 10.2. Informative References

[I-D.ietf-rats-ar4si] Voit, E., Birkholz, H., Hardjono, T., Fossati, T., and V. Scarlata, "Attestation Results for Secure Interactions", Work in Progress, Internet-Draft, draft-ietf-rats-ar4si-03, 6 September 2022, <<https://www.ietf.org/archive/id/draft-ietf-rats-ar4si-03.txt>>.

[I-D.ietf-rats-architecture] Birkholz, H., Thaler, D., Richardson, M., Smith, N., and W. Pan, "Remote Attestation Procedures Architecture", Work in Progress, Internet-Draft, draft-ietf-rats-architecture-22, 28 September 2022, <<https://www.ietf.org/archive/id/draft-ietf-rats-architecture-22.txt>>.

[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-17, 22

October 2022, <<https://www.ietf.org/archive/id/draft-ietf-rats-eat-17.txt>>.

**[TLS-Ext-Registry]** IANA, "Transport Layer Security (TLS) Extensions", 15 November 2005, <<https://www.iana.org/assignments/tls-extensiontype-values>>.

**[TLS-Param-Registry]** IANA, "Transport Layer Security (TLS) Parameters", 23 August 2005, <<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. History

RFC EDITOR: PLEASE REMOVE THE THIS SECTION

### A.1. draft-fossati-tls-attestation-02

- \*Focus on the background check model
- \*Added examples
- \*Updated introduction
- \*Moved attestation format-specific content to related drafts.

### A.2. draft-fossati-tls-attestation-01

- \*Added details about TPM attestation

### A.3. draft-fossati-tls-attestation-00

- \*Initial version

## Appendix B. Working Group Information

The discussion list for the IETF TLS working group is located at the e-mail address [tls@ietf.org](mailto:tls@ietf.org). Information on the group and information on how to subscribe to the list is at <https://www1.ietf.org/mailman/listinfo/tls>

Archives of the list can be found at: <https://www.ietf.org/mail-archive/web/tls/current/index.html>

#### **Authors' Addresses**

Hannes Tschofenig  
Arm Limited

Email: [hannes.tschofenig@arm.com](mailto:hannes.tschofenig@arm.com)

Thomas Fossati  
Arm Limited

Email: [Thomas.Fossati@arm.com](mailto:Thomas.Fossati@arm.com)

Paul Howard  
Arm Limited

Email: [Paul.Howard@arm.com](mailto:Paul.Howard@arm.com)

Ionut Mihalcea  
Arm Limited

Email: [Ionut.Mihalcea@arm.com](mailto:Ionut.Mihalcea@arm.com)

Yogesh Deshpande  
Arm Limited

Email: [Yogesh.Deshpande@arm.com](mailto:Yogesh.Deshpande@arm.com)