Remote ATtestation ProcedureS
Internet-Draft
Intended status: Standards Track
Expires: 30 October 2026

L. Xia
W. Jiang
Huawei Technologies
H. Birkholz
Fraunhofer SIT
J. Zhang
H. Labiod
Huawei Technologies France S.A.S.U.

28 April 2026

Integration of Remote Attestation with Key Negotiation and Key Distribution mechanisms
draft-xia-rats-key-negotiation-integration-02

Abstract

This document describes a generic way to integrate Remote Attestation (RA) with key distribution and key negotiation, so that cryptographic keys are only released to or accepted from attested and policy-compliant environments. It defines an attestation-bound key management mechanism that can be applied on top of existing secure channel and key management protocols, and illustrates it with three representative scenarios: public-cloud KMS, end-user to AI data-center communication, and enterprise-operated KMS. A format-agnostic key binding claim is introduced to express the binding between an Attester' s environment, its public keys, and a session identifier, enabling Relying Parties to use Attestation Results as an input to key distribution and key agreement decisions without changing underlying protocols.

About This Document

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

Status information for this document may be found at https://datatracker.ietf.org/doc/draft-xia-rats-key-negotiation-integration/.

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

Source for this draft and an issue tracker can be found at https://github.com/ietf-rats/draft-xia-rats-key-negotiation-integration.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

Remote Attestation (RA) allows a Relying Party (RP) to learn and appraise properties of a remote computing environment before releasing secrets or accepting security-critical state.

This document describes how to use Attestation Results to drive key distribution and key negotiation, so that cryptographic keys are only provided to or accepted from endpoints whose environment satisfies the RP's policy.

The focus is on a lightweight, RA-driven key management mechanism that can be applied on top of existing transport and key exchange protocols.
The same mechanism underlies three representative deployment scenarios: public-cloud KMS delivering keys to attested cloud workloads, end-user to AI data-center communication with attestation-bound end-to-end (E2E) key agreement, and enterprise-operated KMS distributing keys to attested environments running in public clouds.

1.1. Problem and Approach

Deployments increasingly need to ensure that keys are only used in trustworthy environments, for example when releasing application-layer data-encryption keys from a cloud KMS, when establishing E2E session keys between an end user and an AI service, or when an enterprise KMS delivers keys to workloads hosted in a public cloud.

Today, such requirements are often enforced with deployment-specific mechanisms that are hard to automate and reason about.

This document describes a generic mechanism in which an Attestation Result is used as an input to key management decisions.
Two classes of interactions are considered:

* Attestation-bound key distribution, where an RP releases key material or key-protected services to an Attester only if its environment passes appraisal.

* Attestation-bound E2E key agreement, where a client accepts a shared key with a server only if an Attestation Result for the server satisfies the client's policy.

The mechanism is independent of specific Evidence formats, attestation technologies, and key management protocols, and can be instantiated with both the passport and background-check models defined in the RATS architecture [RFC9334].

The resulting keys can then be used by protocols such as TLS (i.e., [I-D.ietf-tls-hybrid-design]), QUIC, IPsec (i.e., [RFC8784]), OHTTP, or application-layer encryption mechanisms without changes to those protocols.

1.2. Relationship to Other RA Work

Several IETF efforts integrate RA into specific protocols or infrastructures.
Examples include attested TLS [I-D.fossati-tls-attestation], RA based on Exported Authenticators [I-D.fossati-tls-exported-attestation], RA over EDHOC [I-D.ietf-lake-ra], and CSR-attestation mechanisms in PKI enrollment [I-D.ietf-lamps-csr-attestation].

These designs focus on how to carry attestation information inside a given protocol and how to bind it to that protocol's authentication keys and session state ("RA inside TLS", "RA inside EDHOC", or "RA inside PKI").

By contrast, this document assumes that Attestation Results are available (via any suitable transport) and describes how RPs use them to control key distribution and key acceptance, independent of the underlying secure channel or enrollment mechanism.

The mechanisms defined here are therefore complementary to the above designs.
For example, a deployment may use RA over EDHOC or attested TLS to establish a secure channel towards an Attester, and then apply the attestation-bound key distribution pattern so that a cloud or enterprise KMS releases application-layer keys only to endpoints that have been successfully attested.

1.3. Out of Scope and Structure

This document does not define new Evidence formats, Attestation Result types, or protocol extensions for TLS, QUIC, EDHOC, IPsec, or PKI.

It specifies generic mechanisms for using Attestation Results in key management and illustrates them with scenarios; protocol- and API-level details remain deployment-specific.

1.4. Requirements Notation

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.

2. Scenarios

This section describes three representative scenarios for integrating Remote Attestation (RA) with key negotiation and key distribution. A generic pattern for attestation-bound key management is first introduced, and then each scenario is shown as a concrete instantiation of this pattern.

2.1. Generic Attestation-bound Key Management Mechanism

This subsection introduces a generic mechanism for integrating RA with key distribution and key negotiation. Two classes of interactions are considered: - attestation-bound key distribution, where a Relying Party (RP) releases key material to an Attester;

* attestation-bound end-to-end (E2E) key agreement, where a client and server derive a shared key and the client uses an Attestation Result when deciding whether to accept it.

The mechanism is independent of the specific Evidence format and key management protocol, and can be realized with both the passport model and the background-check model defined in the RATS architecture. In all cases, an Attestation Result binds cryptographic key material (for example, an identity public key or an ECDHE public key) to an attested environment and is checked against an RP policy before keys are released or accepted.

This document relies on the freshness properties of Evidence and Attestation Results as defined in the RATS architecture. In particular, mechanisms such as nonces, timestamps, and anti-replay state are provided by the underlying RATS protocols and are not redefined here. Relying Parties are expected to configure these RATS mechanisms according to their deployment-specific threat models.

In the key distribution variant (Figure 1 and Figure 2), the RP acts as a key provider and releases key material only if the Attester’s environment satisfies its policy. The Attestation Result contains a binding between the Attester and a public key credential; the RP uses this credential to encrypt keys for delivery, thereby mitigating diversion attacks where keys would otherwise be delivered to a different or non-compliant environment. The passport and background-check realizations differ mainly in whether the Attester or the RP initiates the interaction with the Verifier.

In the E2E key agreement variant (Figure 3 and Figure 4), a client and server use a key agreement protocol such as ECDHE to derive a shared key, and the client acts as the RP. The Verifier issues an Attestation Result that binds the server’s key agreement parameters (for example, its ECDHE public key) to an attested environment. The client only accepts the resulting shared key if the Attestation Result satisfies its policy, thus ensuring that E2E keys are negotiated with a server whose environment has been successfully attested. The shared secret derived from the key agreement is a *tentative* value until the Client successfully verifies the Attestation Result that binds the server’s ECDHE public key (pk_server) to the attested environment identified by the Client’s policy. The Client MUST NOT derive traffic keys or send application data using this secret prior to successful attestation verification, in order to avoid denial-of-service attacks where an attacker forces the Client to perform expensive key schedule computations without ever providing a valid Attestation Result.

To correlate Evidence, Attestation Results and subsequent key delivery or key acceptance decisions, it is RECOMMENDED to use a session identifier (session_id) that is carried across the protocol interactions. The session_id allows the Attester, Verifier and Relying Party to associate Evidence, Attestation Results and key management messages belonging to the same attestation and key management transaction, and helps mitigate replay or mix-and-match attacks.

A Relying Party MUST only use an Attestation Result for the key distribution or key agreement transaction whose session_id matches the value carried in that Attestation Result. Relying Parties SHOULD NOT reuse an Attestation Result across different session_id values. An implementation SHOULD ensure that Attestation Results are consumed within a time window that is consistent with the freshness properties provided by the underlying RATS mechanisms.

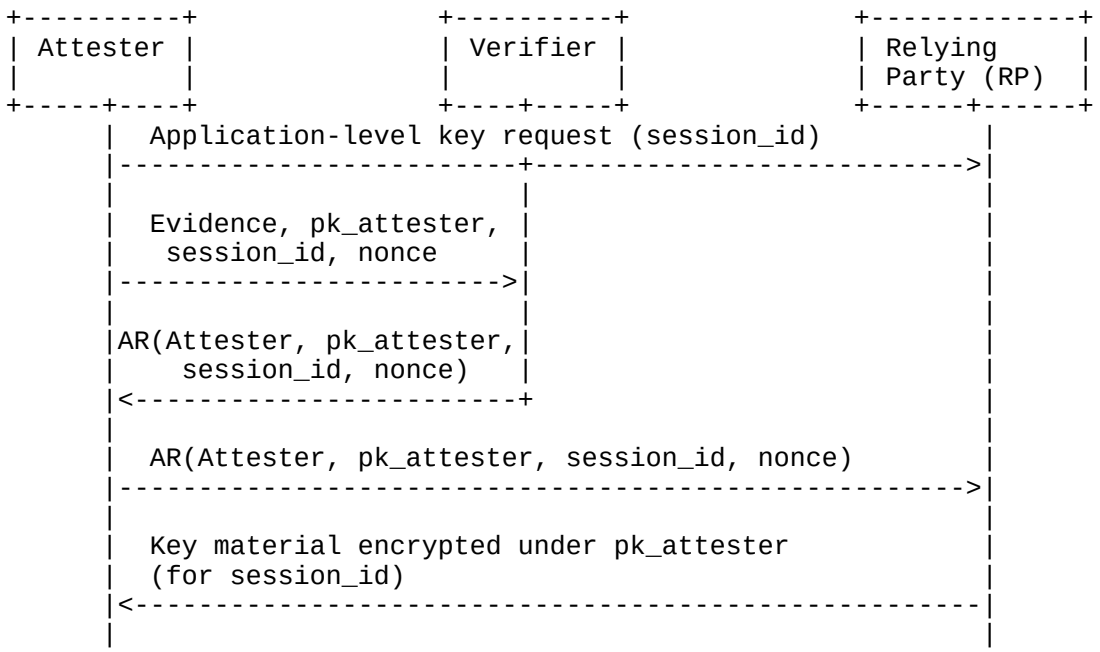

Figure 1: Figure 1: Passport Model Attestation-Bound Key Distribution

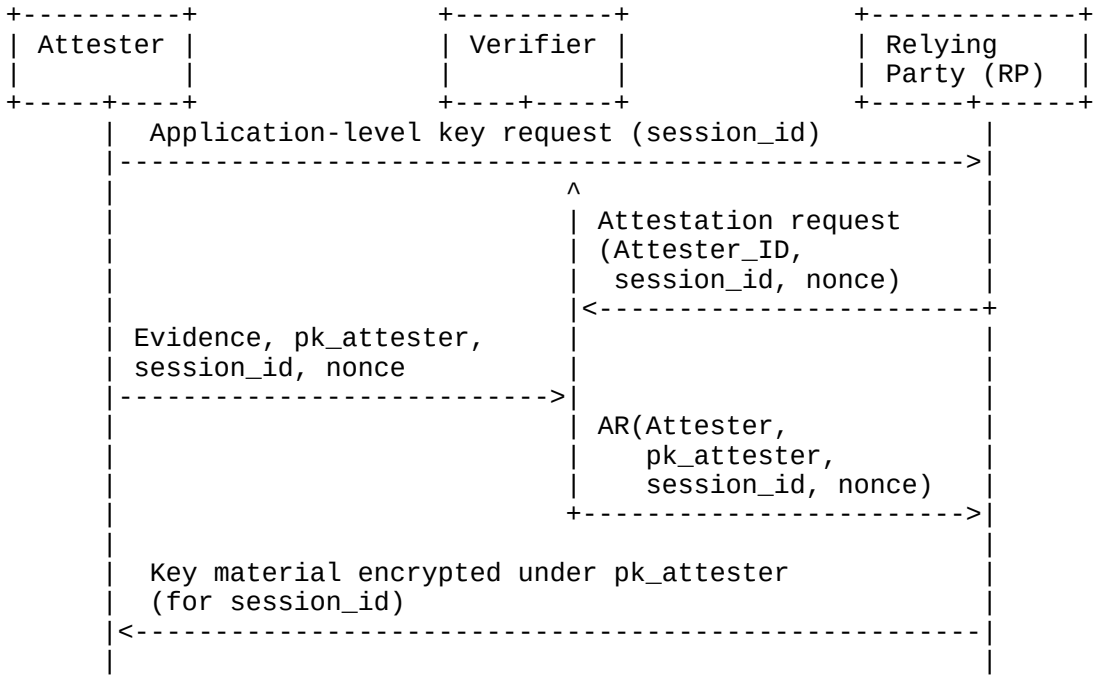

Figure 2: Figure 2 Background-check model: Attestation-Bound Key Distribution

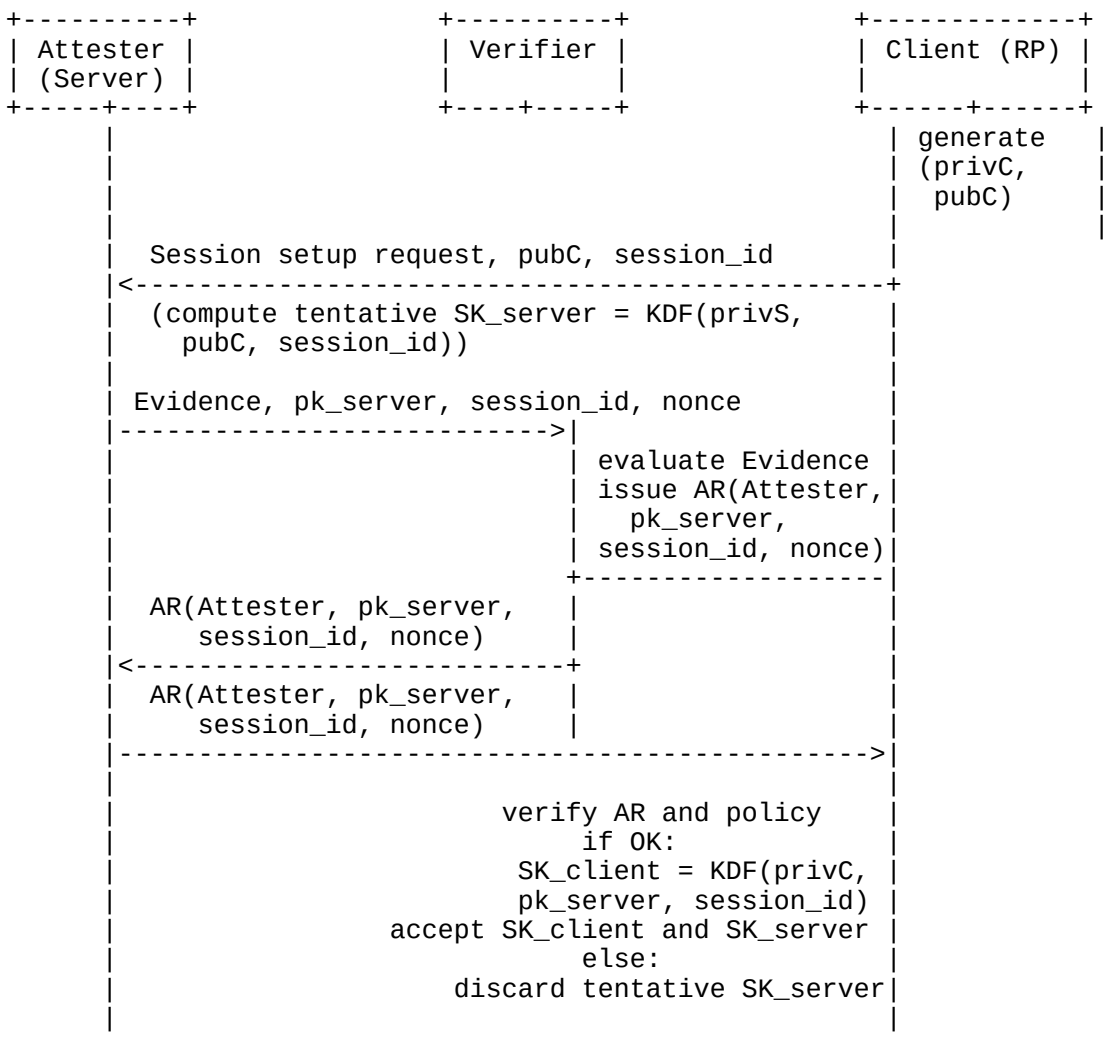

Figure 3: Figure 3 Passport model: Attestation-Bound ECDHE Key Agreement

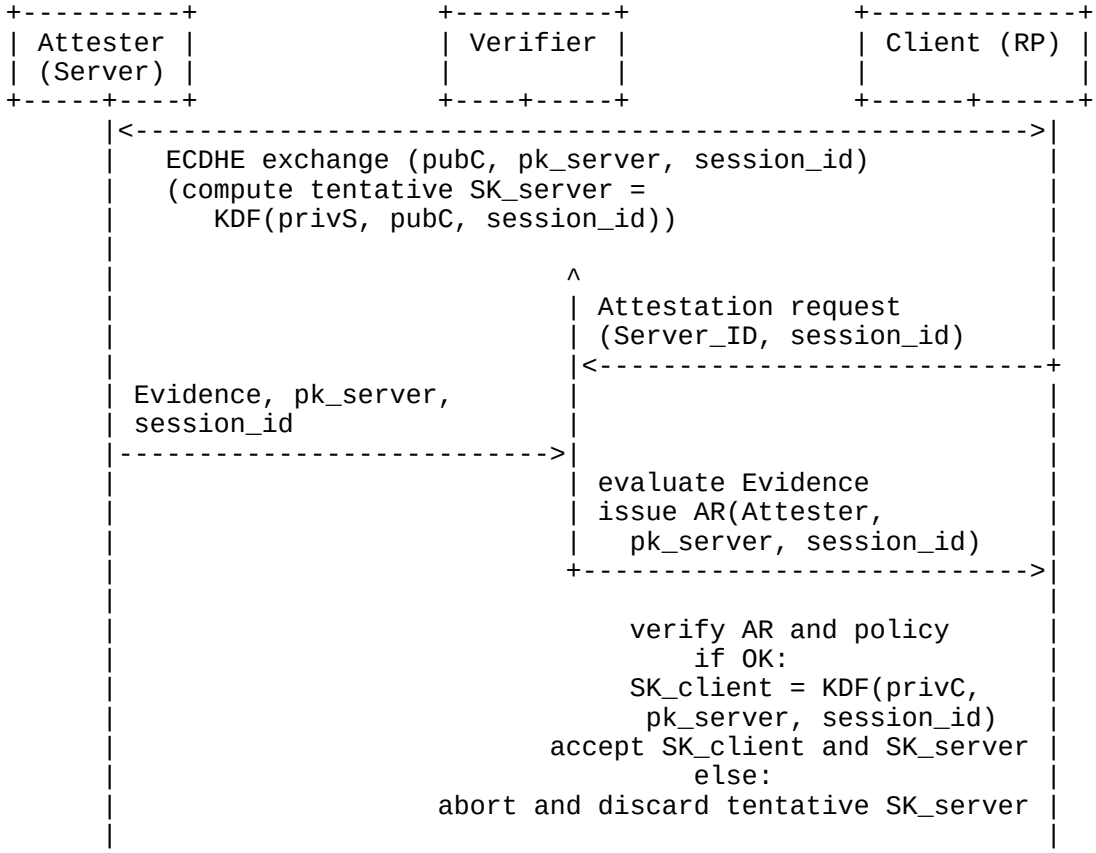

Figure 4: Figure 4 Background-check model: Attestation-Bound ECDHE Key Agreement

Notes: pk_attester A public key credential associated with the Attester and bound in the Attestation Result. It can be a long-term identity key or an ephemeral key, and is used by the RP to protect key delivery in the key distribution pattern.

pk_server The server-side public key used for key agreement (for example, the server’s ECDHE public key). The Attestation Result binds pk_server to the Attester’s environment, and the Client uses this binding when deciding whether to accept the derived session key.

session_id A session identifier carried across Evidence, Attestation Result and key management messages to correlate messages that belong to the same attestation and key management transaction.

nonce A challenge value used by the underlying RATS protocols to guarantee the freshness of Evidence and Attestation Results. The generation, processing and anti-replay handling of nonce values are defined by the RATS mechanisms in use and are not specified in this document.

Application-level key request A request from the Attester to the RP to obtain key material for a specific purpose (for example, an application-layer data key) in the context of a given session_id.

RA request A message that initiates a remote attestation interaction between the Attester and the Verifier and refers to the session_id of the current transaction.

AR(Attester, pk_, session_id) An Attestation Result issued by the Verifier that contains its appraisal about a specific Attester (including an identifier for the Attester) and that binds the Attester to a public key (pk_attester or pk_server) and to a session_id.

2.2. Public Cloud KMS Scenario

The first scenario considers key distribution in a public cloud KMS. In this scenario, the integration of RA and KMS key distribution instantiates the attestation-bound key distribution mechanism described in Section 2.1, with the cloud KMS acting as the RP, the cloud workload as the Attester, and a cloud-operated Verifier. This corresponds to the passport-model message flow illustrated in Figure 1; a background-check realization following the structure of Figure 2 is also possible.

Tenants in a public cloud/compute cluster deploy workloads, they need to first verify the security of their runtime environment through remote attestation before requesting the Key Management Service (KMS) to assign application-layer data keys to the virtual machine (VM)/compute node on which the workload is running. These keys are then used for application-layer data encryption, such as file encryption or disk encryption, rather than for any secure channel protocols. For example, to protect AI/ML model parameters from leakage and tampering, model weights and other parameters are encrypted before transmitting and loading the model to a Trusted Execution Environment (TEE) to ensure that only a TEE that passes attestation and is authorized can obtain the decryption keys.

The Key Management Service (KMS) for public cloud networks includes the root of trust, secure channels between Attester (node, VM, container, service, application) and the KMS, full lifecycle management of keys (including key generation, storage, rotation, and destruction), hierarchical encryption architecture (such as Envelope Encryption), and access control mechanisms. Specifically:

* The basic process for symmetric key distribution and usage is as follows: When an application requires data to be encrypted and shared, it requests the KMS to distribute the DEK (Data Encryption Key) and the EDEK (Encrypted DEK, encrypted using the Customer Master Key (CMK)) to the application via an API. The application then encrypts the data using the DEK, deletes the DEK from memory, and sends the encrypted ciphertext along with the EDEK to the receiving application. The receiving application then requests the KMS to decrypt the EDEK via an API, retrieves the DEK to decrypt the data, and then deletes the DEK from memory.

* The basic process of asymmetric key distribution and usage is as follows: When an application needs to encrypt data, it requests the KMS to generate a public-private key pair via an API, and then uses the obtained public key to encrypt the data. The encrypted ciphertext is then sent to the receiving application. The receiving application calls the KMS's decryption API and the KMS uses the corresponding private key internally to decrypt the data and return the plaintext to the receiving application. The private key never leaves the KMS.When an application requires digital signing of data, it requests the KMS to generate a publicprivate key pair via an API. The public key is then distributed to all parties that need to verify the signature. The application then requests the KMS to sign the data using the corresponding private key via an API, and sends the signature result along with the data to the receiving application. The receiving application uses the public key to verify the signature.

In summary, the KMS provides the applications with the required deliverables, which include application-layer encryption/ authentication, symmetric/asymmetric keys, decrypted plaintext and calculated signatures. For simplicity, the term 'keys' is used here to refer to all these deliverables.

By including the Attester's identity (raw public key or certificate) in the messages throughout the remote attestation process and having the Attester (using its attestation Evidence signing key) and Verifier (using its Attestation Result signing key) endorse and sign it, a binding mechanism between the Attester's Attestation Result and its identity is implemented. Subsequently, the KMS can use the identity's public key for key distribution, ensuring that the keys

are distributed to the correct Attester and eliminating the risk of diversion attacks. During key rotation, the KMS can proactively trigger this process to update keys.

Overall, this approach integrates key distribution into the RA process, achieving automation of key distribution and higher security guarantees based on the security state of the Attester endpoint.

A background-check variant can be realized by letting the KMS request Attestation Results from the Verifier and using them for key release decisions, reusing the pattern described in Section 2.1 and the message flow illustrated in Figure 2.

2.3. End User to AI Data Center Scenario

The second scenario considers E2E key agreement between an end user and an AI data center. In this scenario, the integration of RA and DHE (Diffie-Hellman Ephemeral) and ECDHE (Elliptic-Curve Diffie-Hellman Ephemeral) key negotiation instantiates the attestation-bound E2E key agreement pattern described in Section 2.1, with the end user's client as the RP and the AI data center environment as the Attester. The passport-model realization of this pattern is illustrated in Figure 3, while a background-check realization following the structure of Figure 4 is also possible.

The end user/client accesses the online TEE computing environment, submits their data for business processing or large model inference and must ensure the security of the entire computing environment through remote attestation before establishing a secure connection. There are multiple options for the secure channel protocol that can be established for this use case, such as TLS, IPSec, QUIC and OHTTP. The user may only need to complete end-to-end (E2E) key negotiation based on remote attestation. The negotiated key can be used in various ways, depending on which secure protocol or application layer encryption mechanism is used.

The current main implementation mechanisms for E2E key negotiation are Diffie-Hellman Ephemeral (DHE) and Elliptic-Curve Diffie-Hellman Ephemeral (ECDHE). Taking ECDHE as an example, an ECDHE key exchange generally includes the following steps:

When integrating ECDHE key negotiation into RA, the Client includes its ECDHE public key in the attestation request, the Server provides its ECDHE public key and Evidence to the Verifier, and the Verifier issues an Attestation Result that binds the Server to its ECDHE public key, as shown generically in Figure 3. The Client then verifies the Attestation Result and, if it satisfies the Client's policy, uses the bound ECDHE public key to derive the shared key and associates the key with the attested Server identity.

Overall, this approach integrates E2E key negotiation into the RA process and ensures that the Client only accepts keys that are negotiated with an attested and policy-compliant Server.

A background-check variant can be implemented by letting the Client request Attestation Results for the Server from the Verifier and only accepting the E2E key agreement result if the Attestation Result satisfies its policy, as outlined in Section 2.1 and illustrated by the message flow in Figure 4.

2.4. Enterprise KMS Scenario

The third scenario considers key distribution from an enterprise KMS to cloud environments. In this scenario, the integration of RA and key distribution again instantiates the attestation-bound key distribution pattern described in Section 2.1, with the enterprise KMS as the RP and the cloud environment as the Attester. The passport-model realization of this pattern corresponds to the message flow in Figure 1, while a background-check realization can follow the structure of Figure 2.

Enterprises need to deploy data assets, such as data, applications and systems, on public clouds. Some of these assets are especially critical to the enterprise, such as large private model applications. It is therefore essential to ensure the trustworthiness and security of the operating environment. The process involves first uploading the encrypted data assets to the cloud environment and then performing remote attestation on the operating environment. Once the operating environment passes the security verification, the decryption key is distributed to it for loading and running the decrypted data assets. In fact, the key here can also be generalized to refer to the decrypted plaintext and the calculated signatures provided by an enterprise KMS.

This scenario is similar to the public cloud KMS scenario, except that the public cloud is no longer responsible for key distribution. Instead, the enterprise manages the keys itself and completes key distribution after passing the security verification through RA.

By including the Attester's identity (raw public key or certificate) in the messages throughout the RA process and having the Attester and Verifier endorse and sign it, a binding mechanism between the Attester's Attestation Result and its identity is implemented. Subsequently, the enterprise KMS can use the identity's public key for key distribution, ensuring that the keys are distributed to the correct Attester and eliminating the risk of diversion attacks. During key rotation, the enterprise KMS can proactively trigger this process to update keys.

Overall, this approach integrates key distribution into the RA process from the enterprise perspective, achieving automation of key distribution and higher security guarantees based on the security state of the Attester endpoint.

A background-check variant can again be realized by letting the enterprise KMS request Attestation Results from the Verifier and using them when making key release decisions, following the pattern in Section 2.1 and the background-check flow in Figure 2.

3. Key Binding Claims

This section defines a minimal, format-agnostic claim that can be used in Attestation Results to express the binding between cryptographic keys and an attested environment, in the context of a specific attestation and key management transaction.

The intent of this claim is to provide a common semantic for key binding that can be consumed by Relying Parties when making key distribution and key agreement decisions, independent of the underlying RATS protocol, Evidence format, or key management protocol.

3.1. Overview

In the attestation-bound key distribution and key agreement mechanisms described in Section 2.1, the Verifier issues an Attestation Result (AR) that binds a public key to an Attester's environment and to a session identifier. The Relying Party (RP) then uses this binding when deciding whether and how to release key material or accept a negotiated key for that Attester.

To make this binding explicit and interoperable, this document defines a key binding claim that can be included in an Attestation Result. The claim:

* identifies the public key that is being bound;

* indicates how the key is intended to be used (for example, in key distribution or key agreement);

* associates the key with a specific attestation and key management transaction, via a session_id.

The key binding claim is intended to be used together with other attestation claims (for example, claims that describe the Attester's software, hardware, and configuration state) and with the overall appraisal result produced by the Verifier. It does not by itself convey trustworthiness information; it only expresses that the Verifier has established the cryptographic and transactional binding between a key and an attested environment.

3.2. Key Binding Claim Semantics

A key binding claim in an Attestation Result has the following semantics:

The Verifier asserts that a specific public key is under the control of the Attester's environment that was appraised, and that this public key is bound to the attestation and key management transaction identified by a session_id. The Verifier also indicates the intended usage of the key in that transaction.

More precisely:

* The claim identifies a public key associated with the Attester:

  - In the attestation-bound key distribution mechanism, this public key corresponds to pk_attester, that is, the public key credential that the RP uses to protect key delivery towards the Attester (see Figure 1 and Figure 2).

  - In the attestation-bound key agreement mechanism, this public key corresponds to pk_server, that is, the server-side key agreement public key used to derive the shared session key (for example, an ECDHE public key; see Figure 3 and Figure 4).

* The claim associates the public key with a session identifier, session_id, that is carried across Evidence, Attestation Results, and key management messages for the same transaction. This association is used to mitigate replay and mix-and-match attacks, by ensuring that a key binding is only used in the context of the corresponding attestation and key management interaction.

* The claim indicates the intended usage of the key in the current transaction, such as:

  - use of the key as an encryption target for key distribution (for example, encrypting keys to pk_attester); or

  - use of the key as a peer public key in a key agreement protocol (for example, using pk_server in ECDHE).

The key binding claim is asserted by the Verifier, based on its verification of the Evidence and the RATS protocol-specific protection of the public key and session_id (for example, by including them in signed Evidence or in a protected attestation request). The claim therefore expresses that the Verifier has established a cryptographic binding between the Attester's environment, the public key, and the session_id.

The freshness and anti-replay properties of the key binding claim are inherited from the underlying RATS mechanisms (for example, nonces, timestamps, and anti-replay state) and from the protection of the Attestation Result itself. This document does not define additional freshness mechanisms for the key binding claim.

3.3. Information Elements

The key binding claim is conceptually composed of the following information elements:

* Key Type (kb-key-type): identifies the representation of the public key that is being bound. Examples include:

  - a raw public key (for example, a COSE_Key or JOSE JWK representation);

  - an X.509 certificate that contains the public key;

  - a CBOR-encoded certificate (for example, a C509 certificate).

* Key Value or Hash:

  - kb-key-value: the encoded public key object (for example, a DER-encoded certificate, or a CBOR-encoded key); or

  - kb-key-hash: a cryptographic hash of the encoded public key object (for example, a SHA-256 digest).

A deployment may choose to carry either the full key value, only the hash, or both. For example, carrying the full key value may be preferable on first use, while later interactions can use only the hash to reduce message size, as discussed in Section 8.

* Session Identifier (kb-session-id): the session identifier that links the key binding to a specific attestation and key management transaction. The kb-session-id value MUST be identical to the session_id that is carried in the Evidence and in any key management messages for that transaction.

* Key Usage (kb-usage): indicates how the public key is intended to be used in the current transaction. This document defines the following usage values:

  - key-distribution: the key is used by the RP as an encryption or wrapping key for delivering keys or key-protected services to the Attester (for example, pk_attester in the KMS scenarios described in Section 2.2 and Section 2.4).

  - key-agreement: the key is used by the RP as a peer public key in a key agreement protocol (for example, pk_server in the E2E key agreement scenario described in Section 2.3).

Additional usage values MAY be defined by future specifications or deployment-specific profiles.

3.4. CDDL Definition

When the key binding claim is carried in an Entity Attestation Token (EAT) [RFC9711], it is RECOMMENDED to be specified using CDDL in a way that is consistent with the EAT claim definition style. The following CDDL defines a generic structure for the key binding claim:

```
; Key Binding Claim

key-binding-claim = {
  kb-key-type    : kb-key-type-choice,
  ? kb-key-value : bstr, ; Encoded public key (e.g., DER, CBOR)
  ? kb-key-hash :  bstr, ; Hash of kb-key-value (hash algorithm defined by profile)
  kb-session-id  : bstr,          ; Session identifier for this transaction
  kb-usage       : kb-usage-choice
}

kb-key-type-choice = &(
  raw-public-key     : 1,
  x509-certificate   : 2,
  cbor-certificate   : 3
)

kb-usage-choice = &(
  key-distribution   : 1,
  key-agreement      : 2
)
```

The following considerations apply:

* The kb-key-value and kb-key-hash fields are both OPTIONAL at the encoding level, but at least one of them MUST be present. If both are present, they MUST be consistent (that is, kb-key-hash MUST be the hash of kb-key-value according to the algorithm and encoding agreed in the deployment).

* The concrete encoding of the public key in kb-key-value (for example, the specific COSE_Key, JOSE JWK, X.509, or C509 format) is outside the scope of this document and is determined by deployment-specific or protocol-specific profiles.

* The kb-session-id value MUST be treated as an opaque byte string by the Verifier and RP. Its format and generation are defined by the RATS protocol and key management protocol in use, as discussed in Section 2.1.

* The kb-usage field enables the RP to distinguish between keys that are meant for key distribution and those that are meant for key agreement. A single Attestation Result MAY contain multiple key binding claims with different kb-usage values, for example when an Attester uses different keys for key distribution and key agreement in the same environment.

When the key binding claim is carried in other Attestation Result formats, an equivalent structure and semantics SHOULD be provided by the corresponding specification or profile.

3.5. Relying Party Requirements

An RP that relies on key binding claims for key distribution or key agreement decisions:

* MUST verify that the Attestation Result containing the key binding claim is authentic and has not expired, according to the protection and freshness mechanisms of the RATS protocol and Attestation Result format in use.

* MUST verify that the kb-session-id value in the key binding claim matches the session_id used in the corresponding key management messages for the current transaction. An RP MUST NOT use a key binding claim whose kb-session-id does not match the session_id of the transaction in which the key is to be used.

* MUST verify that the key identified by the key binding claim (either via kb-key-value or kb-key-hash) matches the key that is used in the key management protocol:

  - In the key distribution case, the RP MUST ensure that it encrypts or wraps keys to the public key indicated in the key binding claim (kb-usage = key-distribution).

  - In the key agreement case, the RP MUST ensure that the peer public key used in the key agreement protocol matches the public key indicated in the key binding claim (kb-usage = key-agreement).

* MUST combine the key binding claim with the overall appraisal result and any other relevant claims in the Attestation Result when making key distribution or key acceptance decisions. In particular, the presence of a key binding claim does not by itself imply that the Attester’s environment is trustworthy; it only asserts the binding between a key and that environment.

An RP SHOULD NOT reuse a key binding claim across different transactions or session_id values. If an RP wishes to support re-use of previously established key bindings (for example, for optimization purposes), it SHOULD do so in a way that is consistent with the freshness, anti-replay, and policy requirements of the deployment, and SHOULD be guided by the recommendations in Section 8

4. Remote Attestation Protocol and Message Extensions

This document focuses on defining a generic attestation-bound key management mechanism and the minimal key binding claim semantics needed to support it (see Section 2.1 and Section 3). It does not, in this revision, specify concrete protocol mappings or message extensions for any particular RATS protocol or attestation transport. Future revisions of this document, or separate companion documents, may define how the key binding claim and related information are conveyed in specific RATS protocols and deployment architectures (e.g., by profiling existing attestation transports or by defining new protocol extensions). Until such mappings are specified, the details of how Evidence and Attestation Results carry the key binding claim remain deployment-specific.

TBD.

5. Implementation Status

// RFC Editor: please remove this section prior to publication. This section records the status of known implementations of the protocol defined by this specification at the time of posting of this Internet-Draft, and is based on a proposal described in [RFC7942]. The description of implementations in this section is intended to assist the IETF in its decision processes in progressing drafts to RFCs. Please note that the listing of any individual implementation here does not imply endorsement by the IETF. Furthermore, no effort has been spent to verify the information presented here that was supplied by IETF contributors. This is not intended as, and must not be construed to be, a catalog of available implementations or their features. Readers are advised to note that other implementations may exist.

According to [RFC7942], "this will allow reviewers and working groupsto assign due consideration to documents that have the benefit of running code, which may serve as evidence of valuable experimentation and feedback that have made the implemented protocols more mature. It is up to the individual working groups to use this information as they see fit".

5.1. Trustee

Responsible Organisation: Trustee (open source project within the Confidential Containers).

Location: https://github.com/confidential-containers/trustee

Description: Trustee contains tools and components for attesting confidential guests and providing secrets to them. Collectively, these components are known as Trustee. Trustee typically operates on behalf of the guest owner and interact remotely with guest components. Trustee components include: - Key Broker Service: The KBS is a server that facilitates remote attestation and secret delivery. Its role is similar to that of the Relying Party in the RATS model; - Attestation Service: The AS verifies TEE evidence. In the RATS model this is a Verifier; - Reference Value Provider Service: The RVPS manages reference values used to verify TEE evidence; - KBS Client Tool: This is a simple tool which can be used to test or configure the KBS and AS.

There are two main ways to deploy Trustee: with Docker Compose, on Kubernetes.

Level of Maturity: This is a proof-of-concept prototype implementation.

License: Apache-2.0.

Coverage: This implementation covers most of the aspects of the use case 1 and 3 of this draft.

Contact: Ding Ma, xynnn@linux.alibaba.com

6. Security Considerations

6.1. Key Diversion Attacks

An adversary may attempt a key-parameter misbinding attack by intercepting an Attestation Result and forwarding it to a different endpoint, or by substituting a public key in the Attestation Result for one under the adversary’s control. The mechanisms in this document mitigate this threat by requiring the Verifier to cryptographically bind the Attester’s public key (pk_attester or pk_server) to the Attestation Result (see Section 2.1 and Section 3). A Relying Party MUST verify that the public key used in the actual key distribution or key agreement operation matches the key bound in the Attestation Result, as specified in Section 3.5. If this verification fails, the RP MUST abort the transaction and MUST NOT release any key material. This requirement addresses the threat described in [Misbinding-RPK-TLS].

6.2. Mix-and-Match Attacks

A _mix-and-match attack_ occurs when an adversary combines Attestation Results and key material from different sessions or transactions, for example by replaying a valid Attestation Result from session A in the context of session B, or by substituting the key material from one attested interaction into another. The session_id carried in Evidence, Attestation Results, and key management messages (see Section 2.1) is the primary countermeasure: a Relying Party MUST verify that the kb-session-id value in the key binding claim matches the session_id of the current transaction, and MUST NOT accept an Attestation Result whose session_id does not match (see Section 3.5). Relying Parties SHOULD NOT reuse Attestation Results across different session_id values. The freshness and anti-replay properties of the underlying RATS mechanisms (e.g., nonces, timestamps) further limit the window in which such attacks can be mounted.

6.3. Scope of Attestation and Policy Enforcement

The key binding claim defined in Section 3 asserts only that the Verifier has established a cryptographic binding between a public key and an attested environment. It does not by itself imply that the Attester’s environment is trustworthy or that any particular security policy has been satisfied. A Relying Party MUST evaluate the full Attestation Result — including the overall appraisal outcome and any additional claims about the Attester’s software, hardware, and configuration state — before releasing key material or accepting a negotiated key. In particular, the kb-usage field (Section 3.3) only identifies the intended cryptographic role of the key; it is the RP’s responsibility to ensure that the Attester’s environment satisfies its deployment-specific security policy before keys are distributed or accepted. Implementations SHOULD follow the principle of least privilege: key material SHOULD only be released to environments that satisfy the minimum-required policy, and the RP SHOULD reject requests from environments that report degraded or unknown trust status.

7. Privacy Considerations

The mechanisms defined in this document introduce several interactions that may affect the privacy of Attesters and end users. This section identifies the main privacy concerns and provides recommendations for mitigating them.

7.1. Identity Binding and Attester Identification

The key binding claim defined in Section 3 explicitly binds a public key (pk_attester or pk_server) to an Attester's environment and conveys this binding in an Attestation Result. When a long-term identity key (e.g., a persistent raw public key or an X.509 certificate) is used as kb-key-value, the Attester's identity becomes directly observable to any party that receives the Attestation Result, including the Verifier and the Relying Party.

A static, long-term public key used across multiple attestation transactions enables correlation of those transactions by any observer with access to the Attestation Results or the key management messages. This is a particular concern in the End User to AI Data Center scenario (Section 2.3), where persistent binding of a user's identity key across sessions may allow the Relying Party or an intermediate Verifier to build a profile of the user's attestation history.

To mitigate this, deployments SHOULD prefer the use of short-lived or session-specific public keys (e.g., ephemeral ECDHE keys as pk_server) over long-term identity keys wherever the protocol and policy permit. When a long-term key must be used for pk_attester (e.g., in key distribution scenarios where stable identity is required by the RP policy), the scope of its disclosure SHOULD be limited to the parties that operationally require it. Attestation Results containing long-term identity keys SHOULD NOT be forwarded to parties beyond those involved in the key management transaction.

Implementations are RECOMMENDED to use pseudonymous or unlinkable attestation credentials (e.g., DAA-based attestation) where available and consistent with deployment policy, in order to prevent the Verifier or Relying Party from tracking individual Attesters across independent transactions.

7.2. Hash-based Optimization and Key Traceability

The optimization described in Section 8 allows an Attester to replace the full public key or certificate with its hash (kb-key-hash) in subsequent attestation transactions with the same Relying Party, once the full key has been established in an initial interaction. While this optimization reduces message size, it introduces a privacy implication: the hash of a stable, long-term public key is itself a stable, pseudonymous identifier for the Attester.

If the same kb-key-hash value appears across multiple sessions or protocol interactions, an observer that has access to Attestation Results or Evidence from different sessions can correlate those

sessions to the same Attester, even without access to the full public key. This applies to any party on the path between the Attester, Verifier, and Relying Party that can observe the Attestation Result content.

To mitigate this, deployments that employ the hash optimization SHOULD rotate the underlying public key at a frequency that is consistent with their privacy requirements. When key rotation occurs, the first interaction after the rotation MUST carry the full kb-key-value rather than only the hash. If the deployment requires strong unlinkability across sessions, the use of short-lived keys (which are inherently different across sessions) is preferable to the hash optimization. Implementations SHOULD NOT use the hash optimization in contexts where session-level unlinkability is a requirement.

7.3. Session Identifier Correlation

The session_id (or kb-session-id) is carried across Evidence, Attestation Results, and key management messages to allow the Attester, Verifier, and Relying Party to correlate messages belonging to the same attestation and key management transaction (see Section 2.1 and Section 3.3). This correlation is necessary for the security properties of the mechanism (in particular, to prevent replay and mix-and-match attacks). However, it also means that any party with access to multiple messages in a transaction can link those messages to the same session.

If the session_id value is reused across multiple distinct attestation transactions — for example, because it is derived from a stable Attester identifier or because the implementation fails to generate a fresh value per transaction — it becomes a persistent identifier that enables cross-session correlation. An observer that can access Attestation Results or Evidence from different transactions sharing the same session_id can link those transactions to the same Attester.

To mitigate this, implementations MUST generate a fresh, unpredictable session_id for each independent attestation and key management transaction. The session_id SHOULD be generated as a cryptographically random value of sufficient entropy (e.g., at least 128 bits) to prevent guessing or enumeration. Relying Parties MUST NOT reuse a session_id value across different transactions, and MUST NOT accept an Attestation Result whose kb-session-id does not match the session_id of the current transaction, as required by Section 3.5.

7.4. Verifier Visibility into Attestation History

In both the passport model and the background-check model, the Verifier receives Evidence that includes the Attester's public key and the session_id. The Verifier therefore has visibility into each attestation transaction and may be able to correlate transactions over time if the same Attester key or session_id generation pattern is observed across multiple interactions.

Deployments SHOULD treat the Verifier as a partially trusted entity with respect to Attester privacy. Where privacy is a priority, deployments SHOULD consider architectures in which the Verifier cannot link multiple attestation requests to the same Attester (for example, by using anonymous attestation mechanisms or by routing attestation requests through privacy-preserving intermediaries). The frequency and timing of attestation requests SHOULD be managed to reduce the amount of behavioral information exposed to the Verifier.

7.5. Disclosure Minimization

Attestation Results and Evidence may carry information about the Attester's hardware, software, and configuration state, beyond the key binding claim defined in this document. Such information can be sensitive and may reveal details about the Attester's deployment environment, software stack, or operational context.

Verifiers SHOULD issue Attestation Results that contain the minimum set of claims necessary for the Relying Party to make its key distribution or key acceptance decision. Relying Parties SHOULD NOT request or retain more attestation information than is required for their operational purpose. Attestation Result formats that support selective disclosure or audience-restricted tokens SHOULD be preferred in privacy-sensitive deployments.

8. Optimization Considerations

For the first connection between the Attester and the Relying Party, embedding the raw public key/certificate inside the Attestation Result is necessary as it simplifies the delivery of the raw public key/certificate from the Attester. For subsequent connections between this Attester and the Relying Party, if the raw public key/certificate remains unchanged, the Attester MAY choose to use the hash of its raw public key/certificate instead in the Evidence/Attestation Result between itself and the Verifier, and only forward the Attestation Result that contains the hash of its raw public key/certificate. At the Relying Party side, it compares this hash with the hash of the locally cached raw public key/certificate, and uses the corresponding raw public key/certificate for this session when it

matches. This reduces the payload carried by the Evidence and the Attestation Result.

9. IANA Considerations

9.1. EAT Claims Registry

This document requests IANA to register the following claim in the "JSON Web Token Claims" registry established by [RFC7519] and in the "CBOR Web Token (CWT) Claims" registry established by [RFC8392], in accordance with the EAT claim registration procedures defined in [RFC9711].

9.1.1. Key Binding Claim

The following claim is defined in Section 3 of this document:

Claim Name	Claim Key	Description
key-binding-claim	TBD	Binds a public key and session identifier to an attested environment

Change Controller: IETF Reference: This document

IANA is requested to allocate a claim key for "key-binding-claim" from the "Specification Required" range of the CBOR Web Token (CWT) Claims registry.

9.2. Key Binding Claim Sub-Registries

9.2.1. Key Binding Key Type (kb-key-type) Registry

IANA is requested to create a new registry titled "RATS Key Binding Key Types" under the "Remote ATtestation ProcedureS (RATS)" registry group.

Registration policy: Specification Required [RFC8126].

Initial registrations:

Value	Name	Description
1	raw-public-key	Raw public key (e.g., COSE_Key or JOSE JWK encoding)
2	x509-certificate	X.509 certificate containing the public key
3	cbor-certificate	CBOR-encoded certificate (e.g., C509)

Values 0 and 4127 are available for assignment by IANA. Values 128255 are reserved for Private Use.

9.2.2. Key Binding Key Usage (kb-usage) Registry

IANA is requested to create a new registry titled "RATS Key Binding Key Usages" under the "Remote ATtestation ProcedureS (RATS)" registry group.

Registration policy: Specification Required [RFC8126].

Initial registrations:

Value	Name	Description
1	key-distribution	Key is used as encryption/wrapping target for RA-bound key distribution
2	key-agreement	Key is used as peer public key in RA-bound key agreement (e.g., ECDHE)

Values 0 and 3127 are available for assignment by IANA. Values 128255 are reserved for Private Use.

Additional "kb-usage" values MAY be defined by future specifications or deployment-specific profiles through the Specification Required process defined above.

10. References

10.1. Normative References

[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>.

[RFC7519] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Token (JWT)", RFC 7519, DOI 10.17487/RFC7519, May 2015, <https://www.rfc-editor.org/rfc/rfc7519>.

[RFC7942] Sheffer, Y. and A. Farrel, "Improving Awareness of Running Code: The Implementation Status Section", BCP 205, RFC 7942, DOI 10.17487/RFC7942, July 2016, <https://www.rfc-editor.org/rfc/rfc7942>.

[RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <https://www.rfc-editor.org/rfc/rfc8126>.

[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>.

[RFC8392] Jones, M., Wahlstroem, E., Erdtman, S., and H. Tschofenig, "CBOR Web Token (CWT)", RFC 8392, DOI 10.17487/RFC8392, May 2018, <https://www.rfc-editor.org/rfc/rfc8392>.

[RFC8784] Fluhrer, S., Kampanakis, P., McGrew, D., and V. Smyslov, "Mixing Preshared Keys in the Internet Key Exchange Protocol Version 2 (IKEv2) for Post-quantum Security", RFC 8784, DOI 10.17487/RFC8784, June 2020, <https://www.rfc-editor.org/rfc/rfc8784>.

[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>.

[RFC9711] Lundblade, L., Mandyam, G., O'Donoghue, J., and C. Wallace, "The Entity Attestation Token (EAT)", RFC 9711, DOI 10.17487/RFC9711, April 2025, <https://www.rfc-editor.org/rfc/rfc9711>.

10.2. Informative References

[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.fossati-tls-exported-attestation]
Fossati, T., Sardar, M. U., Reddy.K, T., Sheffer, Y., Tschofenig, H., and I. Mihalcea, "Remote Attestation with Exported Authenticators", Work in Progress, Internet-Draft, draft-fossati-tls-exported-attestation-02, 3 July 2025, <https://datatracker.ietf.org/doc/html/draft-fossati-tls-exported-attestation-02>.

[I-D.ietf-lake-ra]
Song, Y. and G. Selander, "Remote attestation over EDHOC", Work in Progress, Internet-Draft, draft-ietf-lake-ra-05, 26 April 2026, <https://datatracker.ietf.org/doc/html/draft-ietf-lake-ra-05>.

[I-D.ietf-lamps-csr-attestation]
Ounsworth, M., Tschofenig, H., Birkholz, H., Wiseman, M., and N. Smith, "Use of Remote Attestation with Certification Signing Requests", Work in Progress, Internet-Draft, draft-ietf-lamps-csr-attestation-24, 27 March 2026, <https://datatracker.ietf.org/doc/html/draft-ietf-lamps-csr-attestation-24>.

[I-D.ietf-tls-hybrid-design]
Stebila, D., Fluhrer, S., and S. Gueron, "Hybrid key exchange in TLS 1.3", Work in Progress, Internet-Draft, draft-ietf-tls-hybrid-design-16, 7 September 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-tls-hybrid-design-16>.

[Misbinding-RPK-TLS]
Moustafa, M., Sethi, M., and T. Aura, "Misbinding Raw Public Keys to Identities in TLS", 2024, <https://arxiv.org/abs/2411.09770>.

Contributors

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

Authors' Addresses

Liang Xia
Huawei Technologies
Email: frank.xialiang@huawei.com

Weiyu Jiang
Huawei Technologies
Email: jiangweiyu1@huawei.com

Henk Birkholz
Fraunhofer SIT
Email: henk.birkholz@ietf.contact

Jun Zhang
Huawei Technologies France S.A.S.U.
Email: junzhang1@huawei.com

Houda Labiod
Huawei Technologies France S.A.S.U.
Email: houda.labiod@huawei.com