

RATS Working Group
Internet-Draft
Intended status: Informational
Expires: January 9, 2021

H. Birkholz
M. Eckel
Fraunhofer SIT
C. Newton
L. Chen
University of Surrey
July 08, 2020

Reference Interaction Models for Remote Attestation Procedures
draft-birkholz-rats-reference-interaction-model-03

Abstract

This document describes interaction models for remote attestation procedures (RATS). Three conveying mechanisms - Challenge/Response, Uni-Directional, and Streaming Remote Attestation - are illustrated and defined. Analogously, a general overview about the information elements typically used by corresponding conveyance protocols are highlighted. Privacy preserving conveyance of Evidence via Direct Anonymous Attestation is elaborated on for each interaction model, individually.

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 January 9, 2021.

Copyright Notice

Copyright (c) 2020 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 Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	3
2. Terminology	3
3. Disambiguation	4
4. Scope and Intent	4
5. Direct Anonymous Attestation	5
5.1. Endorsers	5
5.2. Endorsers for Direct Anonymous Attestation	6
6. Normative Prerequisites	6
7. Generic Information Elements	7
8. Interaction Models	9
8.1. Challenge/Response Remote Attestation	10
8.2. Uni-Directional Remote Attestation	11
8.3. Streaming Remote Attestation	13
9. Additional Application-Specific Requirements	15
9.1. Confidentiality	15
9.2. Mutual Authentication	15
9.3. Hardware-Enforcement/Support	15
10. Implementation Status	15
10.1. Implementer	16
10.2. Implementation Name	16
10.3. Implementation URL	16
10.4. Maturity	16
10.5. Coverage and Version Compatibility	16
10.6. License	16
10.7. Implementation Dependencies	16
10.8. Contact	17
11. Security and Privacy Considerations	17
12. Acknowledgments	17
13. Change Log	17
14. References	19
14.1. Normative References	19
14.2. Informative References	20
Appendix A. CDDL Specification for a simple CoAP Challenge/Response Interaction	20
Authors' Addresses	21

1. Introduction

Remote Attestation procedures (RATS, [I-D.ietf-rats-architecture]) are workflows composed of roles and interactions, in which Verifiers create Attestation Results about the trustworthiness of an Attester's system component characteristics. The Verifier's assessment in the form of Attestation Results is created based on Attestation Policies and Evidence - trustable and tamper-evident Claims Sets about an Attester's system component characteristics - created by an Attester. The roles `_Attester_` and `_Verifier_`, as well as the Conceptual Messages `_Evidence_` and `_Attestation Results_` are terms defined by the RATS Architecture [I-D.ietf-rats-architecture]. This documents captures interaction models that can be used in specific RATS-related solution documents. The primary focus of this document is the conveyance of attestation Evidence. Specific goals of this document are to:

- o prevent inconsistencies in descriptions of these interaction models in other documents (due to text cloning over time),
- o enable to highlight an exact delta/divergence between the core set of characteristics captured here in this document and variants of these interaction models used in other specifications or solutions, and to
- o illustrate the application of Direct Anonymous Attestation (DAA) for each of the interaction models described.

In summary, this document enables the specification and design of trustworthy and privacy preserving conveyance methods for attestation Evidence from an Attester to a Verifier. While the conveyance of other Conceptual Messages is out-of-scope the methods described can also be applied to the conveyance of Endorsements or Attestation Results.

2. Terminology

This document uses the terms, roles, and concepts defined in [I-D.ietf-rats-architecture]:

Attester, Verifier, Relying Party, Conceptual Message, Evidence, Endorsement, Attestation Result, Appraisal Policy, Attesting Environment, Target Environment

A PKIX Certificate is an X.509v3 format certificate as specified by [RFC5280].

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. Disambiguation

The term "Remote Attestation" is a common expression and often associated or connoted with certain properties. The term "Remote" in this context does not necessarily refer to a remote entity in the scope of network topologies or the Internet. It rather refers to a decoupled system or entities that exchange the payload of the Conceptual Message type called Evidence [I-D.ietf-rats-architecture]. This conveyance can also be "local", if the Verifier is part of the same entity as the Attester, e.g., separate system components of a Composite Device (a single RATS Entity). Examples of these types of co-located environments include: a Trusted Execution Environment (TEE), Baseboard Management Controllers (BMCs), as well as other physical or logical protected/isolated/shielded Computing Environments (e.g. embedded Secure Elements (eSE) or Trusted Platform Modules (TPM)).

4. Scope and Intent

This document focuses on generic interaction models between Attesters and Verifiers in order to convey Evidence. Complementary procedures, functions, or services that are required for a complete semantic binding of the concepts defined in [I-D.ietf-rats-architecture] are out-of-scope of this document. Examples include: identity establishment, key distribution and enrollment, time synchronization, as well as certificate revocation.

Furthermore, any processes and duties that go beyond carrying out remote attestation procedures are out-of-scope. For instance, using the results of a remote attestation that are created by the Verifier, e.g., how to triggering remediation actions or recovery processes, as well as such remediation actions and recovery processes themselves, are also out-of-scope.

The interaction models illustrated in this document are intended to provide a stable basis and reference for other solutions documents inside or outside the IETF. Solution documents of any kind can reference the interaction models in order to avoid text clones and to avoid the danger of subtle discrepancies. Analogously, deviations from the generic model descriptions in this document can be illustrated in solutions documents to highlight distinct contributions.

5. Direct Anonymous Attestation

DAA [DAA] is a signature scheme used in RATS that allows preservation of the privacy of users that are associated with an Attester (e.g. its owner). Essentially, DAA can be seen as a group signature scheme with the feature that given a DAA signature no-one can find out who the signer is, i.e., the anonymity is not revocable. To be able to sign anonymously an Attester has to obtain a credential from a DAA Issuer. The DAA Issuer uses a private/public key pair to generate a credential for an Attester and makes the public key (in the form of a public key certificate) available to the verifier to enable them to validate the DAA signature obtained as part of the Evidence.

In order to support these DAA signatures, the DAA Issuer MUST associate a single key pair with each group of Attesters and use the same key pair when creating the credentials for all of the Attesters in this group. The DAA Issuer's public key certificate for the group replaces the Attester Identity documents in the verification of the Evidence (instead of unique Attester Identity documents). This is in contrast to intuition that there has to be a unique Attester Identity per device.

This document extends the duties of the Endorser role as defined by the RATS architecture with respect to the provision of these Attester Identity documents to Attesters. The existing duties of the Endorser role and the duties of a DAA Issuer are quite similar as illustrated in the following subsections.

5.1. Endorsers

Via its Attesting Environments, an Attester can only create Evidence about its Target Environments. After being appraised to be trustworthy, a Target Environment may become a new Attesting Environment in charge of creating Evidence for further Target Environments. [I-D.ietf-rats-architecture] explains this as Layered Attestation. Layered Attestation has to start with an initial Attesting Environment (i.e., there cannot be turtles all the way down [turtles]). At this rock bottom of Layered Attestation, the Attesting Environments are called Roots of Trust (RoT). An Attester cannot create Evidence about its own RoTs by design. As a consequence, a Verifier requires trustable statements about this subset of Attesting Environments from a different source than the Attester itself. The corresponding trustable statements are called Endorsements and originate from external, trustable entities that take on the role of an Endorser (e.g., supply chain entities).

5.2. Endorsers for Direct Anonymous Attestation

In order to enable DAA to be used, an Endorser role takes on the duties of a DAA Issuer in addition to its already defined duties. DAA Issuers offer zero-knowledge proofs based on public key certificates used for a group of Attesters [DAA]. Effectively, these certificates share the semantics of Endorsements. The differences are:

- o The associated private keys are used by the DAA Issuer to provide an Attester with a credential that it can use to convince the Verifier that its Evidence is valid. To keep their anonymity the Attester randomises this credential each time that it is used.
- o The Verifier can use the DAA Issuer's public key certificate, together with the randomised credential from the Attester, to confirm that the Evidence comes from a valid Attester.
- o A credential is conveyed from an Endorser to an Attester together with the transfer of the public key certificates from Endorser to Verifier.

The zero-knowledge proofs required cannot be created by an Attester alone - like the Endorsements of RoTs - and have to be created by a trustable third entity - like an Endorser. Due to that vast semantic overlap (XXX-mcr:explain), an Endorser in this document can convey trustable third party statements both to a Verifier and an Attester.

6. Normative Prerequisites

In order to ensure an appropriate conveyance of Evidence, the following set of prerequisites MUST be in place to support the implementation of interaction models:

Attester Identity: The provenance of Evidence with respect to a distinguishable Attesting Environment MUST be correct and unambiguous.

An Attester Identity MAY be a unique identity, it MAY be included in a zero-knowledge proof (ZKP), or it MAY be part of a group signature, or it MAY be a randomised DAA credential.

Attestation Evidence Authenticity: Attestation Evidence MUST be correct and authentic.

In order to provide proofs of authenticity, Attestation Evidence SHOULD be cryptographically associated with an identity document (e.g. an PKIX certificate or trusted key material, or a randomised

DAA credential), or SHOULD include a correct and unambiguous and stable reference to an accessible identity document.

Authentication Secret: An Authentication Secret MUST be available exclusively to an Attester's Attesting Environment.

The Attester MUST protect Claims with that Authentication Secret, thereby proving the authenticity of the Claims included in Evidence. The Authentication Secret MUST be established before RATS can take place.

Evidence Freshness: Evidence MUST include an indicator about its Freshness that can be understood by a Verifier. Analogously, interaction models MUST support the conveyance of proofs of freshness in a way that is useful to Verifiers and their appraisal procedures.

Evidence Protection: Evidence MUST be a set of well-formatted and well-protected Claims that an Attester can create and convey to a Verifier in a tamper-evident manner.

7. Generic Information Elements

This section defines the information elements that are vital to all kinds interaction models. Varying from solution to solution, generic information elements can be either included in the scope of protocol messages or can be included in their payload. Ultimately, the following information elements are required by any kind of scalable remote attestation procedure using one or more of the interaction models provided.

Attester Identity ('attesterIdentity'): _mandatory_

A statement about a distinguishable Attester made by an Endorser without accompanying evidence about its validity - used as proof of identity.

In DAA the Attester's identity is not revealed to the verifier. The Attester is issued with a credential by the Endorser that is randomised and then used to anonymously confirm the validity of their evidence. The evidence is verified using the Endorser's public key.

Authentication Secret IDs ('authSecID'): _mandatory_

A statement representing an identifier list that MUST be associated with corresponding Authentication Secrets used to protect Evidence. In DAA, Authentication Secret IDs are

represented by the Endorser (DAA issuer)'s public key that MUST be used to create DAA credentials for the corresponding Authentication Secrets used to protect Evidence.

Each Authentication Secret is uniquely associated with a distinguishable Attesting Environment. Consequently, an Authentication Secret ID also identifies an Attesting Environment. In DAA an Authentication Secret ID does not identify a unique Attesting Environment but associated with a group of Attesting Environments. This is because an Attesting Environment should not be distinguishable and the DAA credential which represents the Attesting Environment is randomised each time it used.

Handle ('handle'): _mandatory_

A statement that is intended to uniquely distinguish received Evidence and/or determine the Freshness of Evidence.

A Verifier can also use a Handle as an indicator for authenticity or attestation provenance, as only Attesters and Verifiers that are intended to exchange Evidence should have knowledge of the corresponding Handles. Examples include Nonces or signed timestamps.

Claims ('claims'): _mandatory_

Claims are assertions that represent characteristics of an Attester's Target Environment.

Claims are part Conceptual Message and are, for example, used to appraise the integrity of Attesters via a Verifiers. The other information elements in this section can be expressed as Claims in any type of Conceptual Messages.

Reference Claims ('refClaims') _mandatory_

Reference Claims are a specific subset of Appraisal Policies as defined in [I-D.ietf-rats-architecture].

Reference Claims are used to appraise the Claims received from an Attester via appraisal by direct comparison. For example, Reference Claims MAY be Reference Integrity Measurements (RIM) or assertions that are implicitly trusted because they are signed by a trusted authority (see Endorsements in [I-D.ietf-rats-architecture]). Reference Claims typically represent (trusted) Claim sets about an Attester's intended platform operational state.

Claim Selection ('claimSelection'): _optional_

A statement that represents a (sub-)set of Claims that can be created by an Attester.

Claim Selections can act as filters that can specify the exact set of Claims to be included in Evidence. An Attester MAY decide whether or not to provide all Claims as requested via a Claim Selection.

Evidence ('signedAttestationEvidence'): _mandatory_

A set of Claims that consists of a list of Authentication Secret IDs that each identifies an Authentication Secret in a single Attesting Environment, the Attester Identity, Claims, and a Handle. Attestation Evidence MUST cryptographically bind all of these information elements. The Evidence MUST be protected via the Authentication Secret. The Authentication Secret MUST be trusted by the Verifier as authoritative.

Attestation Result ('attestationResult'): _mandatory_

An Attestation Result is produced by the Verifier as the output of the appraisal of Evidence. Attestation Results include condensed assertions about integrity or other characteristics of the corresponding Attester.

8. Interaction Models

The following subsections introduce and illustrate the interaction models:

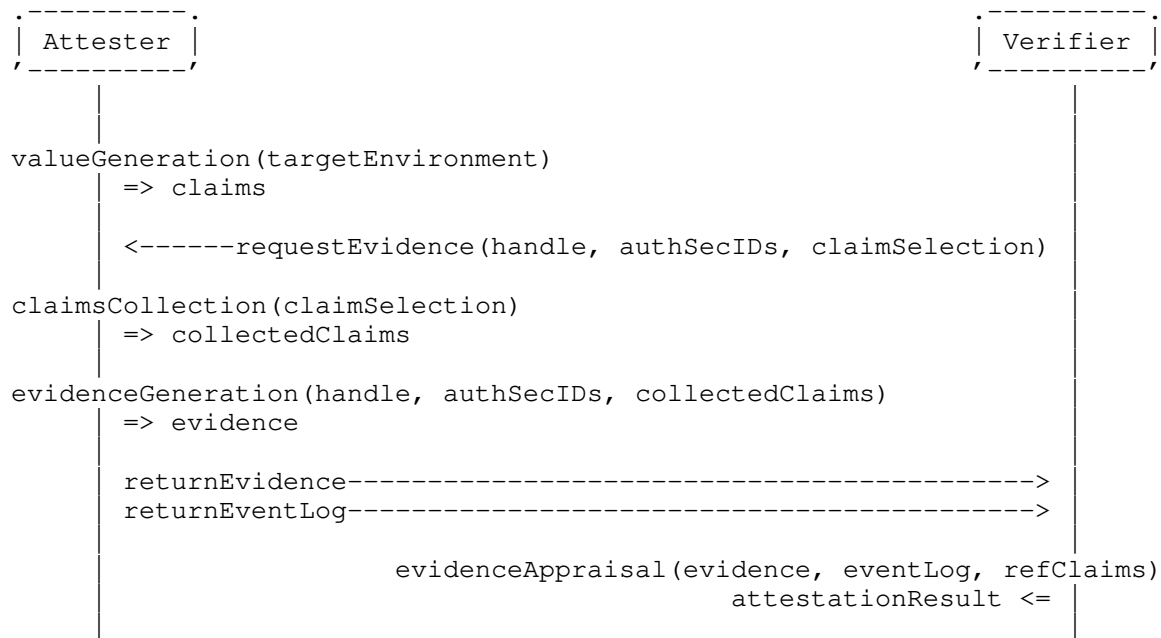
1. Challenge/Response Remote Attestation
2. Uni-Directional Remote Attestation
3. Streaming Remote Attestation

Each section starts with a sequence diagram illustrating the interactions between Attester and Verifier. The other roles RATS roles - mainly Relying Parties and Endorsers - are not relevant for this interaction model. While the interaction models presented focus on the conveyance of Evidence, future work could apply this to the conveyance of other Conceptual Messages, namely Attestation Results, Endorsements, or Appraisal Policies.

All interaction model have a strong focus on the use of a handle to incorporate a proof of freshness. The ways these handles are

processed is the most prominent difference between the three interaction models.

8.1. Challenge/Response Remote Attestation



This Challenge/Response Remote Attestation procedure is initiated by the Verifier, by sending a remote attestation request to the Attester. A request includes a Handle, a list of Authentication Secret IDs, and a Claim Selection.

In the Challenge/Response model, the handle is composed of qualifying data in the form of a cryptographically strongly randomly generated, and therefore unpredictable, nonce. The Verifier-generated nonce is intended to guarantee Evidence freshness.

The list of Authentication Secret IDs selects the attestation keys with which the Attester is requested to sign the Attestation Evidence. Each selected key is uniquely associated with an Attesting Environment of the Attester. As a result, a single Authentication Secret ID identifies a single Attesting Environment.

Analogously, a particular set of Evidence originating from a particular Attesting Environments in a composite device can be requested via multiple Authentication Secret IDs. Methods to acquire

Authentication Secret IDs or mappings between Attesting Environments to Authentication Secret IDs are out-of-scope of this document.

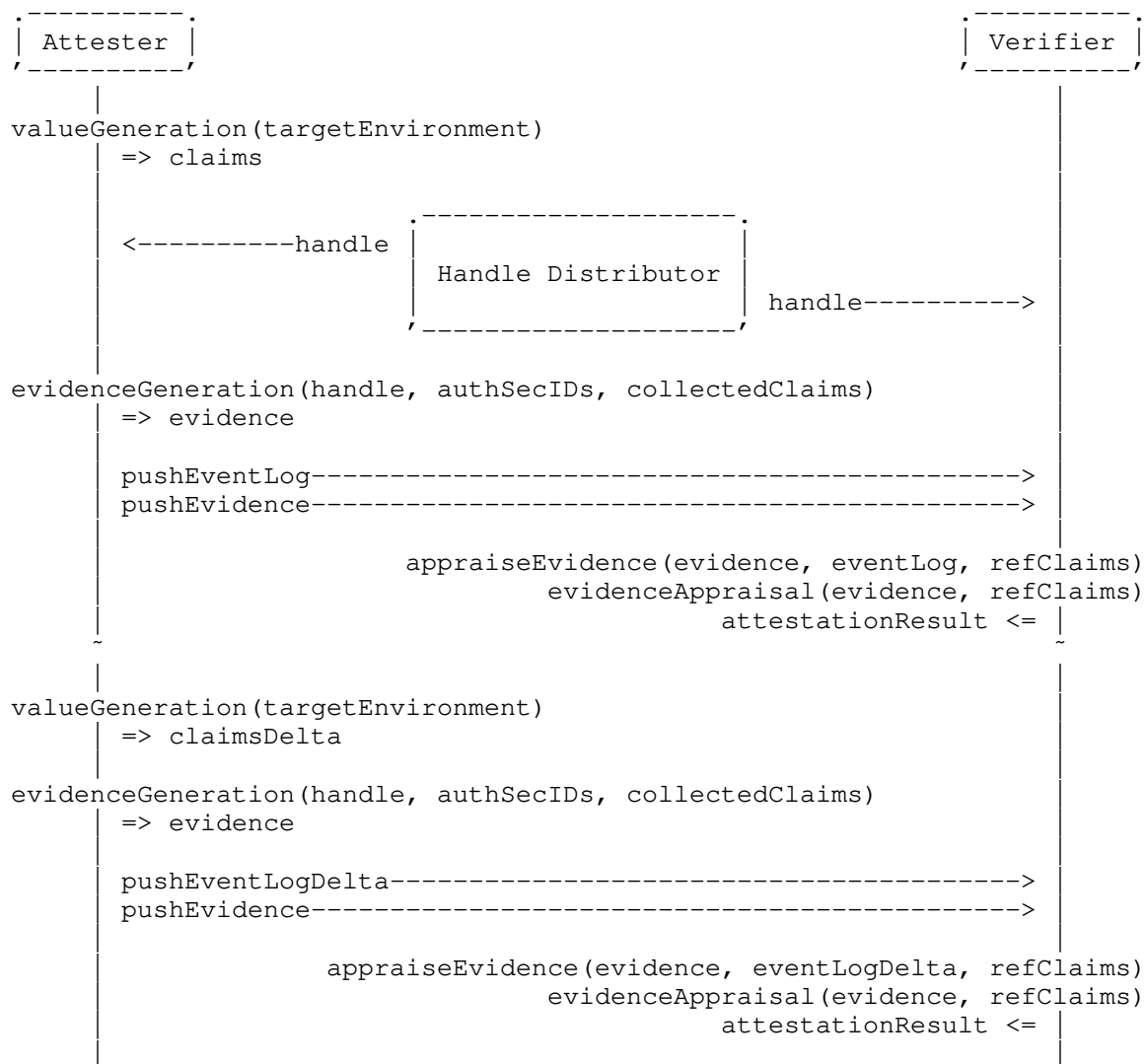
The Claim Selection narrows down the set of Claims collected and used to create Evidence to those that the Verifier requires. If the Claim Selection is omitted, then by default all Claims that are known and available on the Attester MUST be used to create corresponding Evidence. For example when performing a boot integrity evaluation, a Verifier may only be requesting a particular subset of claims about the Attester, such as Evidence about BIOS and firmware the Attester booted up, and not include information about all currently running software.

While it is crucial that Claims, the Handle, as well as the Attester Identity information MUST be cryptographically bound to the signature of Evidence, they may be presented in an encrypted form.

Cryptographic blinding MAY be used at this point. For further reference see section Section 11.

As soon as the Verifier receives signed Evidence, it validates the signature, the Attester Identity, as well as the Nonce, and appraises the Claims. Appraisal procedures are application-specific and can be conducted via comparison of the Claims with corresponding Reference Claims, such as Reference Integrity Measurements. The final output of the Verifier are Attestation Results. Attestation Results constitute new Claims Sets about an Attester's properties and characteristics that enables Relying Parties, for example, to assess an Attester's trustworthiness.

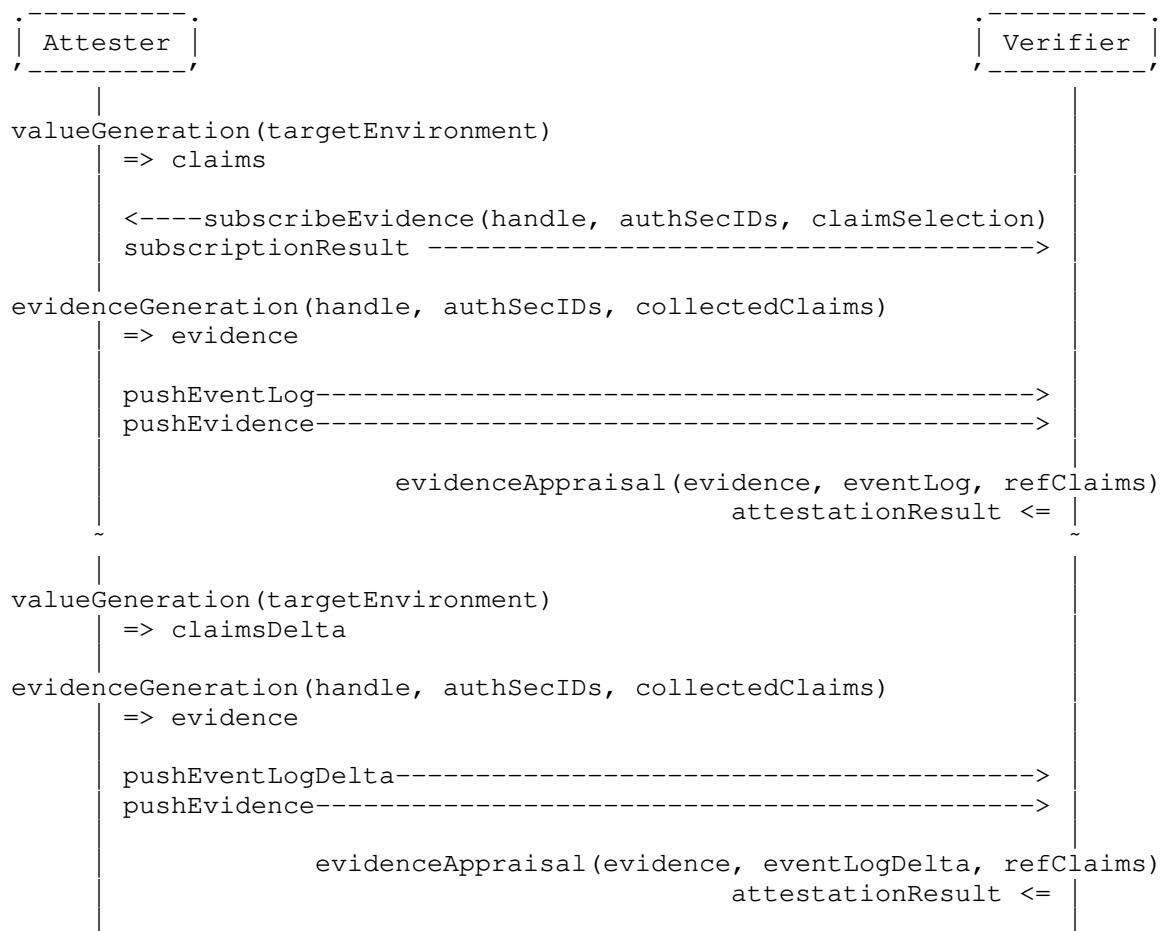
8.2. Uni-Directional Remote Attestation



Uni-Directional Remote Attestation procedures can be initiated both by the Attester and by the Verifier. Initiation by the Attester can result in unsolicited pushes of Evidence to the Verifier. Initiation by the Verifier always results in solicited pushes to the Verifier. The Uni-Directional model uses the same information elements as the Challenge/Response model. In the sequence diagram above, the Attester initiates the conveyance of Evidence (comparable with a RESTful POST operation or the emission of a beacon). While a request of evidence from the Verifier would result in a sequence diagram more similar to the Challenge/Response model (comparable with a RESTful

GET operation), the specific manner how handles are created and used always remains as the distinguishing quality of this model. In the Uni-Directional model, handles are composed of trustable signed timestamps as shown in [I-D.birkholz-rats-tuda], potentially including other qualifying data. The handles are created by an external 3rd entity - the Handle Distributor - that includes a trustworthy source of time and takes on the role of a Time Stamping Authority (TSA, as initially defined in [RFC3161]). Timestamps created from local clocks (absolute clocks using a global timescale, as well as relative clocks, such as tick-counters) of Attesters and Verifiers MUST be cryptographically bound to fresh Handles received from the Handle Distributor. This binding provides a proof of synchronization that MUST be included in every evidence created. Correspondingly, evidence created for conveyance via this model provides a proof that it was fresh at a certain point in time. Effectively, this allows for series of evidence to be pushed to multiple Verifiers, simultaneously. Methods to detect excessive time drift that would mandate a fresh Handle to be received by the Handle Distributor, as well as timing of handle distribution are out-of-scope of this document.

8.3. Streaming Remote Attestation



Streaming Remote Attestation procedures require the setup of subscription state. Setting up subscription state between a Verifier and an Attester is conducted via a subscribe operation. This subscribe operation is used to convey the handles required for Evidence generation. Effectively, this allows for series of evidence to be pushed to a Verifier similar to the Uni-Directional model. While a Handle Distributor is not required in this model, it is also limited to bi-lateral subscription relationships, in which each Verifier has to create and provide its individual handle. Handles provided by a specific subscribing Verifier MUST be used in Evidence generation for that specific Verifier. The Streaming model uses the same information elements as the Challenge/Response and the Uni-Directional model. Methods to detect excessive time drift that would mandate a refreshed Handle to be conveyed via another subscribe operation are out-of-scope of this document.

9. Additional Application-Specific Requirements

Depending on the use cases covered, there can be additional requirements. An exemplary subset is illustrated in this section.

9.1. Confidentiality

Confidentiality of exchanged attestation information may be desirable. This requirement usually is present when communication takes place over insecure channels, such as the public Internet. In such cases, TLS may be used as a suitable communication protocol that preserves confidentiality. In private networks, such as carrier management networks, it must be evaluated whether or not the transport medium is considered confidential.

9.2. Mutual Authentication

In particular use cases mutual authentication may be desirable in such a way that a Verifier also needs to prove its identity to the Attester, instead of only the Attester proving its identity to the Verifier.

9.3. Hardware-Enforcement/Support

Depending on given usage scenarios, hardware support for secure storage of cryptographic keys, crypto accelerators, as well as protected or isolated execution environments can be mandatory requirements. Well-known technologies in support of these requirements are roots of trusts, such as Hardware Security Modules (HSM), Physically Unclonable Functions (PUFs), Shielded Secrets, or Trusted Executions Environments (TEEs).

10. Implementation Status

Note to RFC Editor: Please remove this section as well as references to [BCP205] before AUTH48.

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 [BCP205]. 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 [BCP205], "this will allow reviewers and working groups to 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".

10.1. Implementer

The open-source implementation was initiated and is maintained by the Fraunhofer Institute for Secure Information Technology - SIT.

10.2. Implementation Name

The open-source implementation is named "CHallenge-Response based Remote Attestation" or in short: CHARRA.

10.3. Implementation URL

The open-source implementation project resource can be located via:
<https://github.com/Fraunhofer-SIT/charra>

10.4. Maturity

The code's level of maturity is considered to be "prototype".

10.5. Coverage and Version Compatibility

The current version (commit '847bcde') is aligned with the exemplary specification of the CoAP FETCH bodies defined in section Appendix A of this document.

10.6. License

The CHARRA project and all corresponding code and data maintained on github are provided under the BSD 3-Clause "New" or "Revised" license.

10.7. Implementation Dependencies

The implementation requires the use of the official Trusted Computing Group (TCG) open-source Trusted Software Stack (TSS) for the Trusted Platform Module (TPM) 2.0. The corresponding code and data is also maintained on github and the project resources can be located via:
<https://github.com/tpm2-software/tpm2-tss/>

The implementation uses the Constrained Application Protocol [RFC7252] (<http://coap.technology/>) and the Concise Binary Object Representation [RFC7049] (<https://cbor.io/>).

10.8. Contact

Michael Eckel (michael.eckel@sit.fraunhofer.de)

11. Security and Privacy Considerations

In a remote attestation procedure the Verifier or the Attester MAY want to cryptographically blind several attributes. For instance, information can be part of the signature after applying a one-way function (e. g. a hash function).

There is also a possibility to scramble the Nonce or Attester Identity with other information that is known to both the Verifier and Attester. A prominent example is the IP address of the Attester that usually is known by the Attester itself as well as the Verifier. This extra information can be used to scramble the Nonce in order to counter certain types of relay attacks.

12. Acknowledgments

Olaf Bergmann, Michael Richardson, and Ned Smith

13. Change Log

- o Initial draft -00
- o Changes from version 00 to version 01:
 - * Added details to the flow diagram
 - * Integrated comments from Ned Smith (Intel)
 - * Reorganized sections and
 - * Updated interaction model
 - * Replaced "claims" with "assertions"
 - * Added proof-of-concept CDDL for CBOR via CoAP based on a TPM 2.0 quote operation
- o Changes from version 01 to version 02:

- * Revised the relabeling of "claims" with "assertion" in alignment with the RATS Architecture I-D.
- * Added Implementation Status section
- * Updated interaction model
- * Text revisions based on changes in [I-D.ietf-rats-architecture] and comments provided on rats@ietf.org.
- o Changes from version 02 to version 00 RATS related document
 - * update of the challenge/response diagram
 - * minor rephrasing of Prerequisites section
 - * rephrasing to information elements and interaction model section
- o Changes from version 00 to version 01
 - * added Attestation Authenticity, updated Identity and Secret
 - * relabeled Secret ID to Authentication Secret ID + rephrasing
 - * relabeled Claim Selection to Assertion Selection + rephrasing
 - * relabeled Evidence to (Signed) Attestation Evidence
 - * Added Attestation Result and Reference Assertions
 - * update of the challenge/response diagram and expositional text
 - * added CDDL spec for CoAP FETCH operation proof-of-concept
- o Changes from version 01 to version 02
 - * prepared the inclusion of additional reference models
 - * update to Introduction and Scope section
 - * major update to (Normative) Prerequisites
 - * relabeled Attestation Authenticity to Att. Evidence Authenticity
 - * relabeled Assertion term back to Claim terms

- * added BCP205 Implementation Status section related to Appendix CDDL
- o Changes from version 02 to version 03
 - * major refactoring to now accommodate three interaction models
 - * updated existing and added two new diagrams for models
 - * major refactoring of existing and adding of new diagram description
 - * incorporated content about Direct Anonymous Attestation
 - * integrated comments from Michael Richardson
 - * updated roster

14. References

14.1. Normative References

- [BCP205] 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/info/rfc7942>>.
- [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>>.
- [RFC3161] Adams, C., Cain, P., Pinkas, D., and R. Zuccherato, "Internet X.509 Public Key Infrastructure Time-Stamp Protocol (TSP)", RFC 3161, DOI 10.17487/RFC3161, August 2001, <<https://www.rfc-editor.org/info/rfc3161>>.
- [RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, DOI 10.17487/RFC5280, May 2008, <<https://www.rfc-editor.org/info/rfc5280>>.
- [RFC7049] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", RFC 7049, DOI 10.17487/RFC7049, October 2013, <<https://www.rfc-editor.org/info/rfc7049>>.

- [RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <<https://www.rfc-editor.org/info/rfc7252>>.
- [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/info/rfc8174>>.
- [RFC8610] Birkholz, H., Vigano, C., and C. Bormann, "Concise Data Definition Language (CDDL): A Notational Convention to Express Concise Binary Object Representation (CBOR) and JSON Data Structures", RFC 8610, DOI 10.17487/RFC8610, June 2019, <<https://www.rfc-editor.org/info/rfc8610>>.

14.2. Informative References

- [DAA] Brickell, E., Camenisch, J., and L. Chen, "Direct Anonymous Attestation", ACM Proceedings of the 11rd ACM conference on Computer and Communications Security , page 132-145, 2004.
- [I-D.birkholz-rats-tuda] Fuchs, A., Birkholz, H., McDonald, I., and C. Bormann, "Time-Based Uni-Directional Attestation", draft-birkholz-rats-tuda-02 (work in progress), March 2020.
- [I-D.ietf-rats-architecture] Birkholz, H., Thaler, D., Richardson, M., Smith, N., and W. Pan, "Remote Attestation Procedures Architecture", draft-ietf-rats-architecture-04 (work in progress), May 2020.
- [turtles] Wikipedia, "Turtles all the way down", July 2020, <https://en.wikipedia.org/wiki/Turtles_all_the_way_down>.

Appendix A. CDDL Specification for a simple CoAP Challenge/Response Interaction

The following CDDL specification is an exemplary proof-of-concept to illustrate a potential implementation of the Challenge/Response Interaction Model. The transfer protocol used is CoAP using the FETCH operation. The actual resource operated on can be empty. Both the Challenge Message and the Response Message are exchanged via the FETCH operation and corresponding FETCH Request and FETCH Response body.

In this example, evidence is created via the root-of-trust for reporting primitive operation "quote" that is provided by a TPM 2.0.

RAIM-Bodies = CoAP-FETCH-Body / CoAP-FETCH-Response-Body

```
CoAP-FETCH-Body = [ hello: bool, ; if true, the AK-Cert is conveyed
                    nonce: bytes,
                    pcr-selection: [ + [ tcg-hash-alg-id: uint .size 2, ; TPM2_AL
G_ID
                                [ + pcr: uint .size 1 ],
                                ],
                    ],
```

```
CoAP-FETCH-Response-Body = [ attestation-evidence: TPMS_ATTEST-quote,
                             tpm-native-signature: bytes,
                             ? ak-cert: bytes, ; attestation key certificate
                             ]
```

```
TPMS_ATTEST-quote = [ qualifiediSigner: uint .size 2, ;TPM2B_NAME
                     TPMS_CLOCK_INFO,
                     firmwareVersion: uint .size 8
                     quote-responses: [ * [ pcr: uint .size 1,
                                             + [ pcr-value: bytes,
                                                ? hash-alg-id: uint .size 2,
                                                ],
                                             ],
                                             ? pcr-digest: bytes,
                                             ],
                     ]
```

```
TPMS_CLOCK_INFO = [ clock: uint .size 8,
                    resetCounter: uint .size 4,
                    restartCounter: uint .size 4,
                    save: bool,
                    ]
```

Authors' Addresses

Henk Birkholz
 Fraunhofer SIT
 Rheinstrasse 75
 Darmstadt 64295
 Germany

Email: henk.birkholz@sit.fraunhofer.de

Michael Eckel
Fraunhofer SIT
Rheinstrasse 75
Darmstadt 64295
Germany

Email: michael.eckel@sit.fraunhofer.de

Christopher Newton
University of Surrey

Email: cn0016@surrey.ac.uk

Liqun Chen
University of Surrey

Email: liqun.chen@surrey.ac.uk

RATS Working Group
Internet-Draft
Intended status: Informational
Expires: 12 August 2022

H. Birkholz
Fraunhofer SIT
D. Thaler
Microsoft
M. Richardson
Sandelman Software Works
N. Smith
Intel
W. Pan
Huawei Technologies
8 February 2022

Remote Attestation Procedures Architecture
draft-ietf-rats-architecture-15

Abstract

In network protocol exchanges it is often useful for one end of a communication to know whether the other end is in an intended operating state. This document provides an architectural overview of the entities involved that make such tests possible through the process of generating, conveying, and evaluating evidentiary claims. An attempt is made to provide for a model that is neutral toward processor architectures, the content of claims, and protocols.

Note to Readers

Discussion of this document takes place on the RATS Working Group mailing list (rats@ietf.org), which is archived at <https://mailarchive.ietf.org/arch/browse/rats/> (<https://mailarchive.ietf.org/arch/browse/rats/>).

Source for this draft and an issue tracker can be found at <https://github.com/ietf-rats-wg/architecture> (<https://github.com/ietf-rats-wg/architecture>).

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 12 August 2022.

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	3
2. Reference Use Cases	5
2.1. Network Endpoint Assessment	5
2.2. Confidential Machine Learning Model Protection	6
2.3. Confidential Data Protection	6
2.4. Critical Infrastructure Control	6
2.5. Trusted Execution Environment Provisioning	7
2.6. Hardware Watchdog	7
2.7. FIDO Biometric Authentication	8
3. Architectural Overview	8
3.1. Two Types of Environments of an Attester	10
3.2. Layered Attestation Environments	12
3.3. Composite Device	14
3.4. Implementation Considerations	16
4. Terminology	17
4.1. Roles	17
4.2. Artifacts	18
5. Topological Patterns	19
5.1. Passport Model	19
5.2. Background-Check Model	21
5.3. Combinations	22
6. Roles and Entities	23
7. Trust Model	24
7.1. Relying Party	24
7.2. Attester	25

7.3.	Relying Party Owner	25
7.4.	Verifier	26
7.5.	Endorser, Reference Value Provider, and Verifier Owner .	28
8.	Conceptual Messages	28
8.1.	Evidence	28
8.2.	Endorsements	29
8.3.	Reference Values	29
8.4.	Attestation Results	29
8.5.	Appraisal Policies	31
9.	Claims Encoding Formats	31
10.	Freshness	33
10.1.	Explicit Timekeeping using Synchronized Clocks	33
10.2.	Implicit Timekeeping using Nonces	34
10.3.	Implicit Timekeeping using Epoch IDs	34
10.4.	Discussion	35
11.	Privacy Considerations	36
12.	Security Considerations	37
12.1.	Attester and Attestation Key Protection	37
12.1.1.	On-Device Attester and Key Protection	37
12.1.2.	Attestation Key Provisioning Processes	38
12.2.	Integrity Protection	39
12.3.	Epoch ID-based Attestation	40
12.4.	Trust Anchor Protection	41
13.	IANA Considerations	41
14.	Acknowledgments	41
15.	Notable Contributions	42
16.	Appendix A: Time Considerations	42
16.1.	Example 1: Timestamp-based Passport Model Example	44
16.2.	Example 2: Nonce-based Passport Model Example	45
16.3.	Example 3: Epoch ID-based Passport Model Example	46
16.4.	Example 4: Timestamp-based Background-Check Model Example	47
16.5.	Example 5: Nonce-based Background-Check Model Example .	48
17.	References	49
17.1.	Normative References	49
17.2.	Informative References	49
	Contributors	52
	Authors' Addresses	53

1. Introduction

The question of how one system can know that another system can be trusted has found new interest and relevance in a world where trusted computing elements are maturing in processor architectures.

Systems that have been attested and verified to be in a good state (for some value of "good") can improve overall system posture. Conversely, systems that cannot be attested and verified to be in a good state can be given reduced access or privileges, taken out of service, or otherwise flagged for repair.

For example:

- * A bank back-end system might refuse to transact with another system that is not known to be in a good state.
- * A healthcare system might refuse to transmit electronic healthcare records to a system that is not known to be in a good state.

In Remote Attestation Procedures (RATS), one peer (the "Attester") produces believable information about itself - Evidence - to enable a remote peer (the "Relying Party") to decide whether to consider that Attester a trustworthy peer or not. RATS are facilitated by an additional vital party, the Verifier.

The Verifier appraises Evidence via appraisal policies and creates the Attestation Results to support Relying Parties in their decision process. This document defines a flexible architecture consisting of attestation roles and their interactions via conceptual messages. Additionally, this document defines a universal set of terms that can be mapped to various existing and emerging Remote Attestation Procedures. Common topological patterns and the sequence of data flows associated with them, such as the "Passport Model" and the "Background-Check Model", are illustrated. The purpose is to define useful terminology for remote attestation and enable readers to map their solution architecture to the canonical attestation architecture provided here. Having a common terminology that provides well-understood meanings for common themes such as roles, device composition, topological patterns, and appraisal procedures is vital for semantic interoperability across solutions and platforms involving multiple vendors and providers.

Amongst other things, this document is about trust and trustworthiness. Trust is a choice one makes about another system. Trustworthiness is a quality about the other system that can be used in making one's decision to trust it or not. This is subtle difference and being familiar with the difference is crucial for using this document. Additionally, the concepts of freshness and trust relationships with respect to RATS are elaborated on to enable implementers to choose appropriate solutions to compose their Remote Attestation Procedures.

2. Reference Use Cases

This section covers a number of representative and generic use cases for remote attestation, independent of specific solutions. The purpose is to provide motivation for various aspects of the architecture presented in this document. Many other use cases exist, and this document does not intend to have a complete list, only to illustrate a set of use cases that collectively cover all the functionality required in the architecture.

Each use case includes a description followed by an additional summary of the Attester and Relying Party roles derived from the use case.

2.1. Network Endpoint Assessment

Network operators want trustworthy reports that include identity and version information about the hardware and software on the machines attached to their network. Examples of reports include purposes, such as inventory summaries, audit results, anomaly notifications, typically including the maintenance of log records or trend reports. The network operator may also want a policy by which full access is only granted to devices that meet some definition of hygiene, and so wants to get Claims about such information and verify its validity. Remote attestation is desired to prevent vulnerable or compromised devices from getting access to the network and potentially harming others.

Typically, a trustworthy solution starts with a specific component (sometimes referred to as a root of trust) that often provides trustworthy device identity, and performs a series of operations that enables trustworthiness appraisals for other components. Such components perform operations that help determine the trustworthiness of yet other components, by collecting, protecting or signing measurements. Measurements that have been signed by such components comprise Evidence that when evaluated either supports or refutes a claim of trustworthiness. Measurements can describe a variety of attributes of system components, such as hardware, firmware, BIOS, software, etc.

Attester: A device desiring access to a network.

Relying Party: Network equipment such as a router, switch, or access point, responsible for admission of the device into the network.

2.2. Confidential Machine Learning Model Protection

A device manufacturer wants to protect its intellectual property. The intellectual property's scope primarily encompasses the machine learning (ML) model that is deployed in the devices purchased by its customers. The protection goals include preventing attackers, potentially the customer themselves, from seeing the details of the model.

This typically works by having some protected environment in the device go through a remote attestation with some manufacturer service that can assess its trustworthiness. If remote attestation succeeds, then the manufacturer service releases either the model, or a key to decrypt a model already deployed on the Attester in encrypted form, to the requester.

Attester: A device desiring to run an ML model.

Relying Party: A server or service holding ML models it desires to protect.

2.3. Confidential Data Protection

This is a generalization of the ML model use case above, where the data can be any highly confidential data, such as health data about customers, payroll data about employees, future business plans, etc. As part of the attestation procedure, an assessment is made against a set of policies to evaluate the state of the system that is requesting the confidential data. Attestation is desired to prevent leaking data via compromised devices.

Attester: An entity desiring to retrieve confidential data.

Relying Party: An entity that holds confidential data for release to authorized entities.

2.4. Critical Infrastructure Control

Potentially harmful physical equipment (e.g., power grid, traffic control, hazardous chemical processing, etc.) is connected to a network in support of critical infrastructure. The organization managing such infrastructure needs to ensure that only authorized code and users can control corresponding critical processes, and that these processes are protected from unauthorized manipulation or other threats. When a protocol operation can affect a critical system component of the infrastructure, devices attached to that critical component require some assurances depending on the security context, including that: a requesting device or application has not been

compromised, and the requesters and actors act on applicable policies. As such, remote attestation can be used to only accept commands from requesters that are within policy.

Attester: A device or application wishing to control physical equipment.

Relying Party: A device or application connected to potentially dangerous physical equipment (hazardous chemical processing, traffic control, power grid, etc.).

2.5. Trusted Execution Environment Provisioning

A Trusted Application Manager (TAM) server is responsible for managing the applications running in a Trusted Execution Environment (TEE) of a client device, as described in [I-D.ietf-tee-architecture]. To achieve its purpose, the TAM needs to assess the state of a TEE, or of applications in the TEE, of a client device. The TEE conducts Remote Attestation Procedures with the TAM, which can then decide whether the TEE is already in compliance with the TAM's latest policy. If not, the TAM has to uninstall, update, or install approved applications in the TEE to bring it back into compliance with the TAM's policy.

Attester: A device with a TEE capable of running trusted applications that can be updated.

Relying Party: A TAM.

2.6. Hardware Watchdog

There is a class of malware that holds a device hostage and does not allow it to reboot to prevent updates from being applied. This can be a significant problem, because it allows a fleet of devices to be held hostage for ransom.

A solution to this problem is a watchdog timer implemented in a protected environment such as a Trusted Platform Module (TPM), as described in [TCGarch] section 43.3. If the watchdog does not receive regular, and fresh, Attestation Results as to the system's health, then it forces a reboot.

Attester: The device that should be protected from being held hostage for a long period of time.

Relying Party: A watchdog capable of triggering a procedure that resets a device into a known, good operational state.

2.7. FIDO Biometric Authentication

In the Fast IDentity Online (FIDO) protocol [WebAuthN], [CTAP], the device in the user's hand authenticates the human user, whether by biometrics (such as fingerprints), or by PIN and password. FIDO authentication puts a large amount of trust in the device compared to typical password authentication because it is the device that verifies the biometric, PIN and password inputs from the user, not the server. For the Relying Party to know that the authentication is trustworthy, the Relying Party needs to know that the Authenticator part of the device is trustworthy. The FIDO protocol employs remote attestation for this.

The FIDO protocol supports several remote attestation protocols and a mechanism by which new ones can be registered and added. Remote attestation defined by RATS is thus a candidate for use in the FIDO protocol.

Attester: FIDO Authenticator.

Relying Party: Any web site, mobile application back-end, or service that relies on authentication data based on biometric information.

3. Architectural Overview

Figure 1 depicts the data that flows between different roles, independent of protocol or use case.

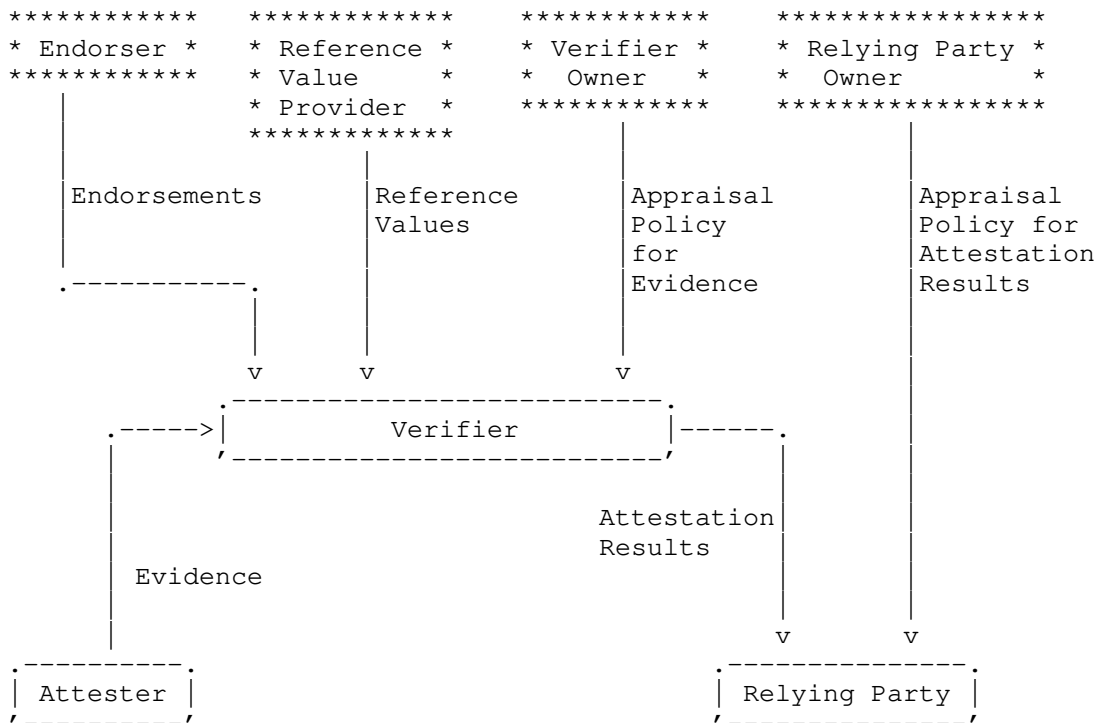


Figure 1: Conceptual Data Flow

The text below summarizes the activities conducted by the roles illustrated in Figure 1. Roles are assigned to entities. Entities are often system components [RFC4949], such as devices. As the term device is typically more intuitive than the term entity or system component, device is often used as a illustrative synonym throughout this document.

The Attester role is assigned to entities that create Evidence that is conveyed to a Verifier.

The Verifier role is assigned to entities that use the Evidence, any Reference Values from Reference Value Providers, and any Endorsements from Endorsers, by applying an Appraisal Policy for Evidence to assess the trustworthiness of the Attester. This procedure is called the appraisal of Evidence.

Subsequently, the Verifier role generates Attestation Results for use by Relying Parties.

The Appraisal Policy for Evidence might be obtained from the Verifier Owner via some protocol mechanism, or might be configured into the Verifier by the Verifier Owner, or might be programmed into the Verifier, or might be obtained via some other mechanism.

The Relying Party role is assigned to entities that uses Attestation Results by applying its own appraisal policy to make application-specific decisions, such as authorization decisions. This procedure is called the appraisal of Attestation Results.

The Appraisal Policy for Attestation Results might be obtained from the Relying Party Owner via some protocol mechanism, or might be configured into the Relying Party by the Relying Party Owner, or might be programmed into the Relying Party, or might be obtained via some other mechanism.

See Section 8 for further discussion of the conceptual messages shown in Figure 1. Section Section 4 provides a more complete definition of all RATS roles.

3.1. Two Types of Environments of an Attester

As shown in Figure 2, an Attester consists of at least one Attesting Environment and at least one Target Environment co-located in one entity. In some implementations, the Attesting and Target Environments might be combined into one environment. Other implementations might have multiple Attesting and Target Environments, such as in the examples described in more detail in Section 3.2 and Section 3.3. Other examples may exist. All compositions of Attesting and Target Environments discussed in this architecture can be combined into more complex implementations.

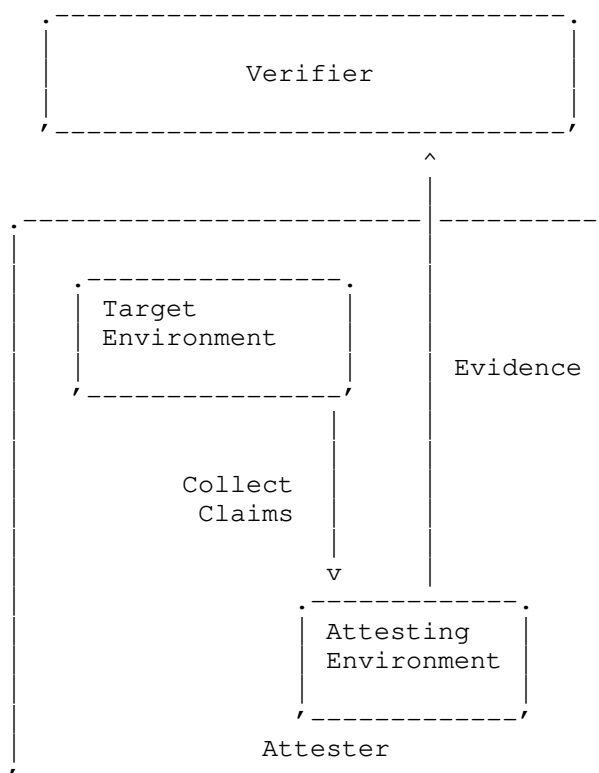


Figure 2: Two Types of Environments

Claims are collected from Target Environments. That is, Attesting Environments collect the values and the information to be represented in Claims, by reading system registers and variables, calling into subsystems, taking measurements on code, memory, or other security related assets of the Target Environment. Attesting Environments then format the Claims appropriately, and typically use key material and cryptographic functions, such as signing or cipher algorithms, to generate Evidence. There is no limit to or requirement on the types of hardware or software environments that can be used to implement an Attesting Environment, for example: Trusted Execution Environments (TEEs), embedded Secure Elements (eSEs), Trusted Platform Modules (TPMs) [TCGarch], or BIOS firmware.

An arbitrary execution environment may not, by default, be capable of Claims collection for a given Target Environment. Execution environments that are designed specifically to be capable of Claims collection are referred to in this document as Attesting Environments. For example, a TPM doesn't actively collect Claims

itself, it instead requires another component to feed various values to the TPM. Thus, an Attesting Environment in such a case would be the combination of the TPM together with whatever component is feeding it the measurements.

3.2. Layered Attestation Environments

By definition, the Attester role generates Evidence. An Attester may consist of one or more nested environments (layers). The bottom layer of an Attester has an Attesting Environment that is typically designed to be immutable or difficult to modify by malicious code. In order to appraise Evidence generated by an Attester, the Verifier needs to trust various layers, including the bottom Attesting Environment. Trust in the Attester's layers, including the bottom layer, can be established in various ways as discussed in Section 7.4.

In layered attestation, Claims can be collected from or about each layer beginning with an initial layer. The corresponding Claims can be structured in a nested fashion that reflects the nesting of the Attester's layers. Normally, Claims are not self-asserted, rather a previous layer acts as the Attesting Environment for the next layer. Claims about an initial layer typically are asserted by an Endorser.

The example device illustrated in Figure 3 includes (A) a BIOS stored in read-only memory, (B) a bootloader, and (C) an operating system kernel.

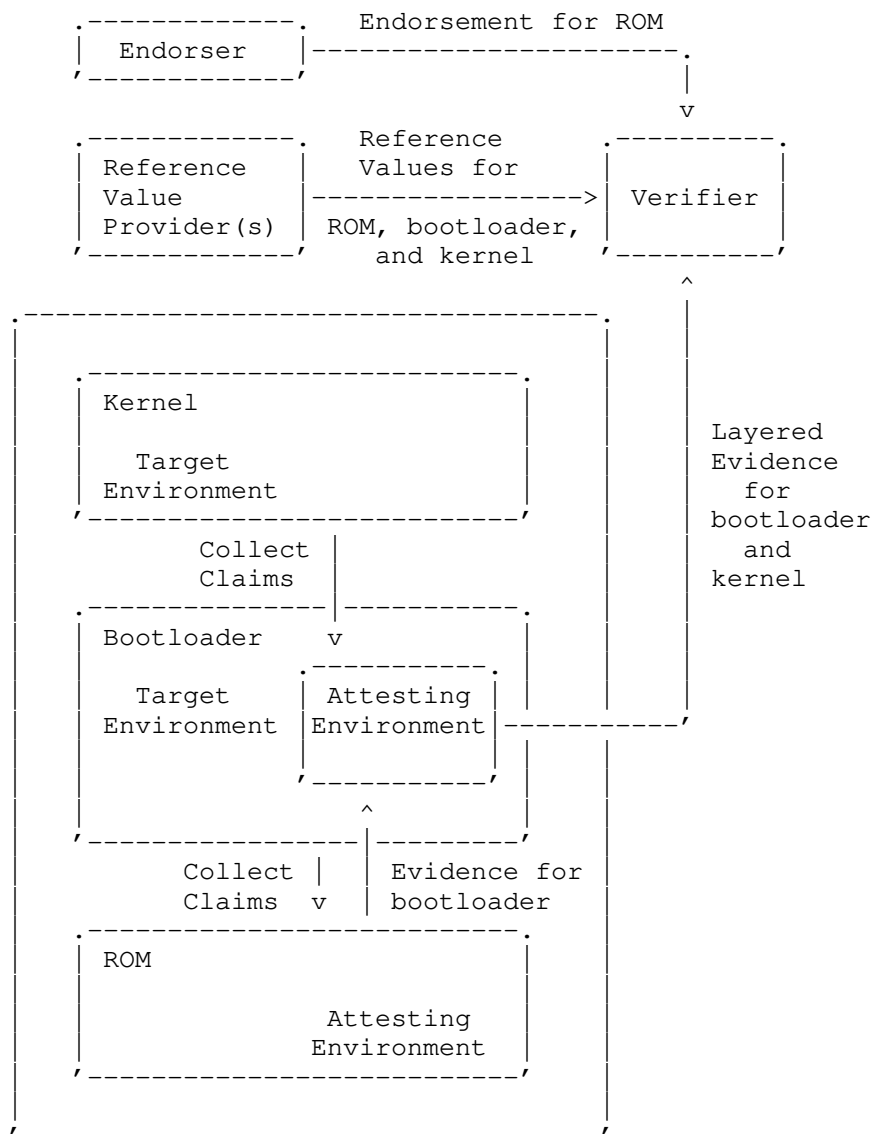


Figure 3: Layered Attester

The first Attesting Environment, the ROM in this example, has to ensure the integrity of the bootloader (the first Target Environment). There are potentially multiple kernels to boot, and the decision is up to the bootloader. Only a bootloader with intact integrity will make an appropriate decision. Therefore, the Claims relating to the integrity of the bootloader have to be measured securely. At this stage of the boot-cycle of the device, the Claims collected typically cannot be composed into Evidence.

After the boot sequence is started, the BIOS conducts the most important and defining feature of layered attestation, which is that the successfully measured bootloader now becomes (or contains) an Attesting Environment for the next layer. This procedure in layered attestation is sometimes called "staging". It is important that the bootloader not be able to alter any Claims about itself that were collected by the BIOS. This can be ensured having those Claims be either signed by the BIOS or stored in a tamper-proof manner by the BIOS.

Continuing with this example, the bootloader's Attesting Environment is now in charge of collecting Claims about the next Target Environment, which in this example is the kernel to be booted. The final Evidence thus contains two sets of Claims: one set about the bootloader as measured and signed by the BIOS, plus a set of Claims about the kernel as measured and signed by the bootloader.

This example could be extended further by making the kernel become another Attesting Environment for an application as another Target Environment. This would result in a third set of Claims in the Evidence pertaining to that application.

The essence of this example is a cascade of staged environments. Each environment has the responsibility of measuring the next environment before the next environment is started. In general, the number of layers may vary by device or implementation, and an Attesting Environment might even have multiple Target Environments that it measures, rather than only one as shown by example in Figure 3.

3.3. Composite Device

A composite device is an entity composed of multiple sub-entities such that its trustworthiness has to be determined by the appraisal of all these sub-entities.

Each sub-entity has at least one Attesting Environment collecting the Claims from at least one Target Environment, then this sub-entity generates Evidence about its trustworthiness. Therefore, each sub-

entity can be called an Attester. Among all the Attesters, there may be only some which have the ability to communicate with the Verifier while others do not.

For example, a carrier-grade router consists of a chassis and multiple slots. The trustworthiness of the router depends on all its slots' trustworthiness. Each slot has an Attesting Environment, such as a TEE, collecting the Claims of its boot process, after which it generates Evidence from the Claims.

Among these slots, only a "main" slot can communicate with the Verifier while other slots cannot. But other slots can communicate with the main slot by the links between them inside the router. So the main slot collects the Evidence of other slots, produces the final Evidence of the whole router and conveys the final Evidence to the Verifier. Therefore the router is a composite device, each slot is an Attester, and the main slot is the lead Attester.

Another example is a multi-chassis router composed of multiple single carrier-grade routers. Multi-chassis router setups create redundancy groups that provide higher throughput by interconnecting multiple routers in these groups, which can be treated as one logical router for simpler management. A multi-chassis router setup provides a management point that connects to the Verifier. Typically one router in the group is designated as the main router. Other routers in the multi-chassis setup are connected to the main router only via physical network links and are therefore managed and appraised via the main router's help. Consequently, a multi-chassis router setup is a composite device, each router is an Attester, and the main router is the lead Attester.

Figure 4 depicts the conceptual data flow for a composite device.

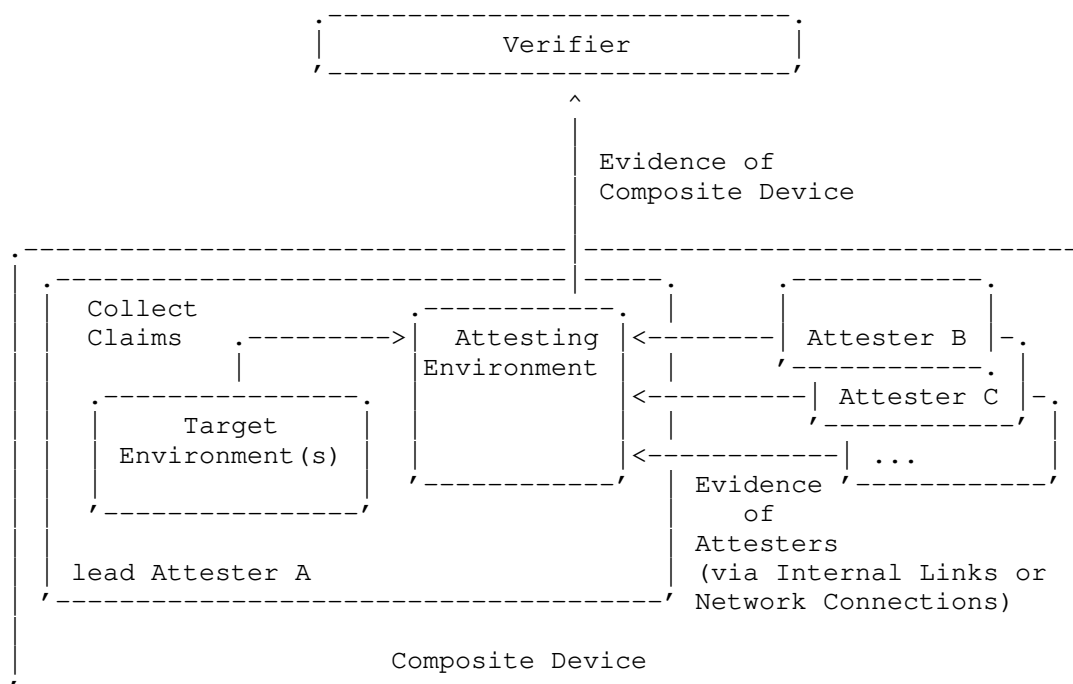


Figure 4: Composite Device

In a composite device, each Attester generates its own Evidence by its Attesting Environment(s) collecting the Claims from its Target Environment(s). The lead Attester collects Evidence from other Attesters and conveys it to a Verifier. Collection of Evidence from sub-entities may itself be a form of Claims collection that results in Evidence asserted by the lead Attester. The lead Attester generates Evidence about the layout of the whole composite device, while sub-Attesters generate Evidence about their respective (sub-)modules.

In this scenario, the trust model described in Section 7 can also be applied to an inside Verifier.

3.4. Implementation Considerations

An entity can take on multiple RATS roles (e.g., Attester, Verifier, Relying Party, etc.) at the same time. Multiple entities can cooperate to implement a single RATS role as well. In essence, the combination of roles and entities can be arbitrary. For example, in the composite device scenario, the entity inside the lead Attester can also take on the role of a Verifier, and the outer entity of

Verifier can take on the role of a Relying Party. After collecting the Evidence of other Attesters, this inside Verifier uses Endorsements and appraisal policies (obtained the same way as by any other Verifier) as part of the appraisal procedures that generate Attestation Results. The inside Verifier then conveys the Attestation Results of other Attesters to the outside Verifier, whether in the same conveyance protocol as part of the Evidence or not.

4. Terminology

This document uses the following terms.

4.1. Roles

Attester: A role performed by an entity (typically a device) whose Evidence must be appraised in order to infer the extent to which the Attester is considered trustworthy, such as when deciding whether it is authorized to perform some operation.

Produces: Evidence

Relying Party: A role performed by an entity that depends on the validity of information about an Attester, for purposes of reliably applying application specific actions. Compare /relying party/ in [RFC4949].

Consumes: Attestation Results, Appraisal Policy for Attestation Results

Verifier: A role performed by an entity that appraises the validity of Evidence about an Attester and produces Attestation Results to be used by a Relying Party.

Consumes: Evidence, Reference Values, Endorsements, Appraisal Policy for Evidence

Produces: Attestation Results

Relying Party Owner: A role performed by an entity (typically an administrator), that is authorized to configure Appraisal Policy for Attestation Results in a Relying Party.

Produces: Appraisal Policy for Attestation Results

Verifier Owner: A role performed by an entity (typically an administrator), that is authorized to configure Appraisal Policy for Evidence in a Verifier.

Produces: Appraisal Policy for Evidence

Endorser: A role performed by an entity (typically a manufacturer) whose Endorsements may help Verifiers appraise the authenticity of Evidence and infer further capabilities of the Attester.

Produces: Endorsements

Reference Value Provider: A role performed by an entity (typically a manufacturer) whose Reference Values help Verifiers appraise Evidence to determine if acceptable known Claims have been recorded by the Attester.

Produces: Reference Values

4.2. Artifacts

Claim: A piece of asserted information, often in the form of a name/value pair. Claims make up the usual structure of Evidence and other RATS artifacts. Compare /claim/ in [RFC7519].

Endorsement: A secure statement that an Endorser vouches for the integrity of an Attester's various capabilities such as Claims collection and Evidence signing.

Consumed By: Verifier

Produced By: Endorser

Evidence: A set of Claims generated by an Attester to be appraised by a Verifier. Evidence may include configuration data, measurements, telemetry, or inferences.

Consumed By: Verifier

Produced By: Attester

Attestation Result: The output generated by a Verifier, typically including information about an Attester, where the Verifier vouches for the validity of the results.

Consumed By: Relying Party

Produced By: Verifier

Appraisal Policy for Evidence: A set of rules that informs how a Verifier evaluates the validity of information about an Attester. Compare /security policy/ in [RFC4949].

Consumed By: Verifier

Produced By: Verifier Owner

Appraisal Policy for Attestation Results: A set of rules that direct how a Relying Party uses the Attestation Results regarding an Attester generated by the Verifiers. Compare /security policy/ in [RFC4949].

Consumed by: Relying Party

Produced by: Relying Party Owner

Reference Values: A set of values against which values of Claims can be compared as part of applying an Appraisal Policy for Evidence. Reference Values are sometimes referred to in other documents as known-good values, golden measurements, or nominal values, although those terms typically assume comparison for equality, whereas here Reference Values might be more general and be used in any sort of comparison.

Consumed By: Verifier

Produced By: Reference Value Provider

5. Topological Patterns

Figure 1 shows a data-flow diagram for communication between an Attester, a Verifier, and a Relying Party. The Attester conveys its Evidence to the Verifier for appraisal, and the Relying Party receives the Attestation Result from the Verifier. This section refines the data-flow diagram by describing two reference models, as well as one example composition thereof. The discussion that follows is for illustrative purposes only and does not constrain the interactions between RATS roles to the presented patterns.

5.1. Passport Model

The passport model is so named because of its resemblance to how nations issue passports to their citizens. The nature of the Evidence that an individual needs to provide to its local authority is specific to the country involved. The citizen retains control of the resulting passport document and presents it to other entities when it needs to assert a citizenship or identity Claim, such as an airport immigration desk. The passport is considered sufficient because it vouches for the citizenship and identity Claims, and it is issued by a trusted authority. Thus, in this immigration desk analogy, the citizen is the Attester, the passport issuing agency is

a Verifier, the passport application and identifying information (e.g., birth certificate) is the Evidence, the passport is an Attestation Result, and the immigration desk is a Relying Party.

In this model, an Attester conveys Evidence to a Verifier, which compares the Evidence against its appraisal policy. The Verifier then gives back an Attestation Result which the Attester treats as opaque data. The Attester does not consume the Attestation Result, but might cache it. The Attester can then present the Attestation Result (and possibly additional Claims) to a Relying Party, which then compares this information against its own appraisal policy.

Three ways in which the process may fail include:

- * First, the Verifier may not issue a positive Attestation Result due to the Evidence not passing the Appraisal Policy for Evidence.
- * The second way in which the process may fail is when the Attestation Result is examined by the Relying Party, and based upon the Appraisal Policy for Attestation Results, the result does not pass the policy.
- * The third way is when the Verifier is unreachable or unavailable.

As with any other information needed by the Relying Party to make an authorization decision, an Attestation Result can be carried in a resource access protocol between the Attester and Relying Party. In this model the details of the resource access protocol constrain the serialization format of the Attestation Result. The format of the Evidence on the other hand is only constrained by the Attester-Verifier remote attestation protocol. This implies that interoperability and standardization is more relevant for Attestation Results than it is for Evidence.

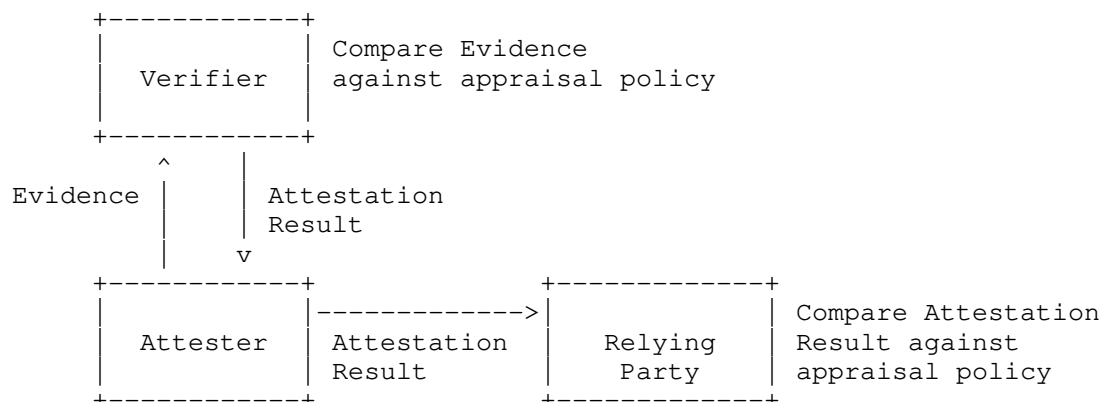


Figure 5: Passport Model

5.2. Background-Check Model

The background-check model is so named because of the resemblance of how employers and volunteer organizations perform background checks. When a prospective employee provides Claims about education or previous experience, the employer will contact the respective institutions or former employers to validate the Claim. Volunteer organizations often perform police background checks on volunteers in order to determine the volunteer's trustworthiness. Thus, in this analogy, a prospective volunteer is an Attester, the organization is the Relying Party, and the organization that issues a report is a Verifier.

In this model, an Attester conveys Evidence to a Relying Party, which treats it as opaque and simply forwards it on to a Verifier. The Verifier compares the Evidence against its appraisal policy, and returns an Attestation Result to the Relying Party. The Relying Party then compares the Attestation Result against its own appraisal policy.

The resource access protocol between the Attester and Relying Party includes Evidence rather than an Attestation Result, but that Evidence is not processed by the Relying Party. Since the Evidence is merely forwarded on to a trusted Verifier, any serialization format can be used for Evidence because the Relying Party does not need a parser for it. The only requirement is that the Evidence can be encapsulated in the format required by the resource access protocol between the Attester and Relying Party.

However, like in the Passport model, an Attestation Result is still consumed by the Relying Party. Code footprint and attack surface area can be minimized by using a serialization format for which the Relying Party already needs a parser to support the protocol between the Attester and Relying Party, which may be an existing standard or widely deployed resource access protocol. Such minimization is especially important if the Relying Party is a constrained node.

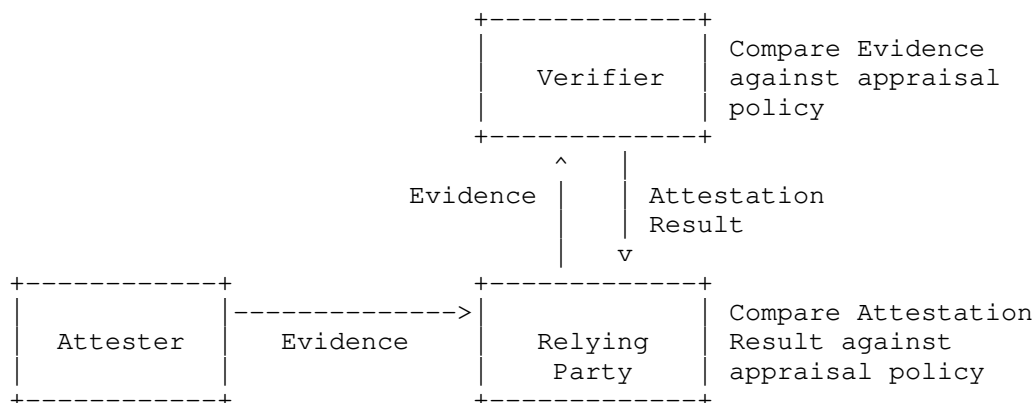


Figure 6: Background-Check Model

5.3. Combinations

One variation of the background-check model is where the Relying Party and the Verifier are on the same machine, performing both functions together. In this case, there is no need for a protocol between the two.

It is also worth pointing out that the choice of model depends on the use case, and that different Relying Parties may use different topological patterns.

The same device may need to create Evidence for different Relying Parties and/or different use cases. For instance, it would use one model to provide Evidence to a network infrastructure device to gain access to the network, and the other model to provide Evidence to a server holding confidential data to gain access to that data. As such, both models may simultaneously be in use by the same device.

Figure 7 shows another example of a combination where Relying Party 1 uses the passport-check model, whereas Relying Party 2 uses an extension of the background-check model. Specifically, in addition to the basic functionality shown in Figure 6, Relying Party 2 actually provides the Attestation Result back to the Attester, allowing the Attester to use it with other Relying Parties. This is the model that the Trusted Application Manager plans to support in the TEEP architecture [I-D.ietf-teep-architecture].

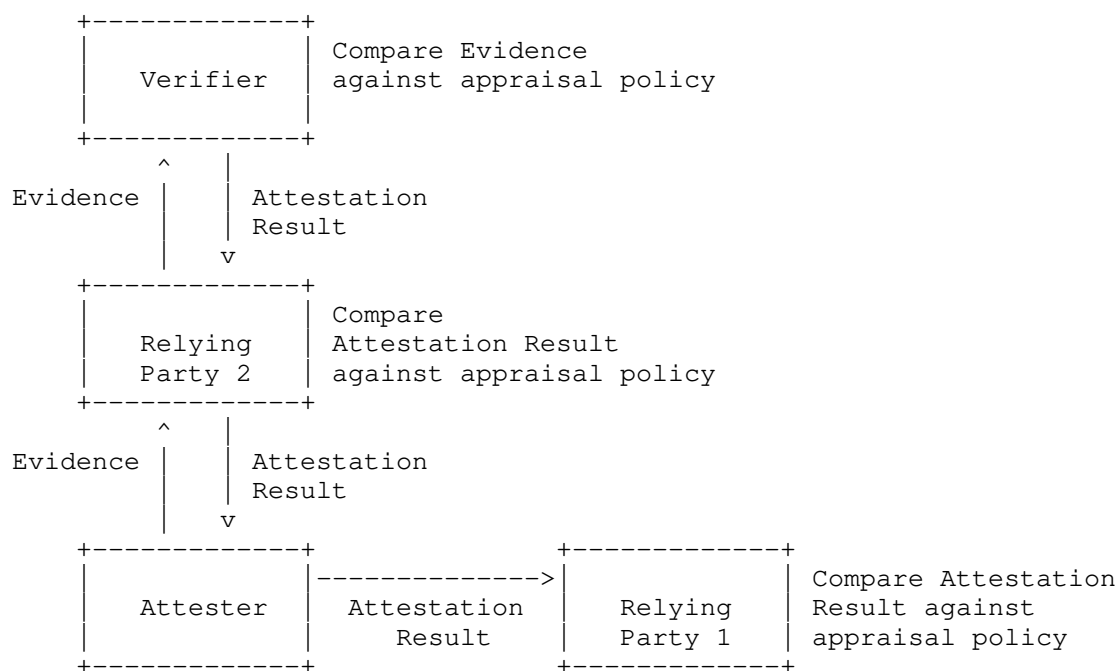


Figure 7: Example Combination

6. Roles and Entities

An entity in the RATS architecture includes at least one of the roles defined in this document.

An entity can aggregate more than one role into itself, such as being both a Verifier and a Relying Party, or being both a Reference Value Provider and an Endorser. As such, any conceptual messages (see Section 8 for more discussion) originating from such roles might also be combined. For example, Reference Values might be conveyed as part of an appraisal policy if the Verifier Owner and Reference Value Provider roles are combined. Similarly, Reference Values might be conveyed as part of an Endorsement if the Endorser and Reference Value Provider roles are combined.

Interactions between roles aggregated into the same entity do not necessarily use the Internet Protocol. Such interactions might use a loopback device or other IP-based communication between separate environments, but they do not have to. Alternative channels to convey conceptual messages include function calls, sockets, GPIO interfaces, local busses, or hypervisor calls. This type of conveyance is typically found in composite devices. Most

importantly, these conveyance methods are out-of-scope of RATS, but they are presumed to exist in order to convey conceptual messages appropriately between roles.

In essence, an entity that combines more than one role creates and consumes the corresponding conceptual messages as defined in this document.

7. Trust Model

7.1. Relying Party

This document covers scenarios for which a Relying Party trusts a Verifier that can appraise the trustworthiness of information about an Attester. Such trust is expressed by storing one or more "trust anchors" in a secure location known as a trust anchor store.

As defined in [RFC6024], "A trust anchor represents an authoritative entity via a public key and associated data. The public key is used to verify digital signatures, and the associated data is used to constrain the types of information for which the trust anchor is authoritative." The trust anchor may be a certificate or it may be a raw public key along with additional data if necessary such as its public key algorithm and parameters. In the context of this document, a trust anchor may also be a symmetric key, as in [TCG-DICE-SIBDA] or the symmetric mode described in [I-D.tschofenig-rats-psa-token].

Thus, trusting a Verifier might be expressed by having the Relying Party store the Verifier's key or certificate in its trust anchor store, or might be expressed by storing the public key or certificate of an entity (e.g., a Certificate Authority) that is in the Verifier's certificate path. For example, the Relying Party can verify that the Verifier is an expected one by out of band establishment of key material, combined with a protocol like TLS to communicate. There is an assumption that between the establishment of the trusted key material and the creation of the Evidence, that the Verifier has not been compromised.

For a stronger level of security, the Relying Party might require that the Verifier first provide information about itself that the Relying Party can use to assess the trustworthiness of the Verifier before accepting its Attestation Results. Such process would provide a stronger level of confidence in the correctness of the information provided, such as a belief that the authentic Verifier has not been compromised by malware.

For example, one explicit way for a Relying Party "A" to establish such confidence in the correctness of a Verifier "B", would be for B to first act as an Attester where A acts as a combined Verifier/Relying Party. If A then accepts B as trustworthy, it can choose to accept B as a Verifier for other Attesters.

Similarly, the Relying Party also needs to trust the Relying Party Owner for providing its Appraisal Policy for Attestation Results, and in some scenarios the Relying Party might even require that the Relying Party Owner go through a remote attestation procedure with it before the Relying Party will accept an updated policy. This can be done similarly to how a Relying Party could establish trust in a Verifier as discussed above, i.e., verifying credentials against a trust anchor store and optionally requiring Attestation Results from the Relying Party Owner.

7.2. Attester

In some scenarios, Evidence might contain sensitive information such as Personally Identifiable Information (PII) or system identifiable information. Thus, an Attester must trust entities to which it conveys Evidence, to not reveal sensitive data to unauthorized parties. The Verifier might share this information with other authorized parties, according to a governing policy that address the handling of sensitive information (potentially included in Appraisal Policies for Evidence). In the background-check model, this Evidence may also be revealed to Relying Party(s).

When Evidence contains sensitive information, an Attester typically requires that a Verifier authenticates itself (e.g., at TLS session establishment) and might even request a remote attestation before the Attester sends the sensitive Evidence. This can be done by having the Attester first act as a Verifier/Relying Party, and the Verifier act as its own Attester, as discussed above.

7.3. Relying Party Owner

The Relying Party Owner might also require that the Relying Party first act as an Attester, providing Evidence that the Owner can appraise, before the Owner would give the Relying Party an updated policy that might contain sensitive information. In such a case, authentication or attestation in both directions might be needed, in which case typically one side's Evidence must be considered safe to share with an untrusted entity, in order to bootstrap the sequence. See Section 11 for more discussion.

7.4. Verifier

The Verifier trusts (or more specifically, the Verifier's security policy is written in a way that configures the Verifier to trust) a manufacturer, or the manufacturer's hardware, so as to be able to appraise the trustworthiness of that manufacturer's devices. Such trust is expressed by storing one or more trust anchors in the Verifier's trust anchor store.

In a typical solution, a Verifier comes to trust an Attester indirectly by having an Endorser (such as a manufacturer) vouch for the Attester's ability to securely generate Evidence through Endorsements (see Section 8.2). Endorsements might describe the ways in which the Attester resists attack, protects secrets and measures Target Environments. Consequently, the Endorser's key material is stored in the Verifier's trust anchor store so that Endorsements can be authenticated and used in the Verifier's appraisal process.

In some solutions, a Verifier might be configured to directly trust an Attester by having the Verifier have the Attester's key material (rather than the Endorser's) in its trust anchor store.

Such direct trust must first be established at the time of trust anchor store configuration either by checking with an Endorser at that time, or by conducting a security analysis of the specific device. Having the Attester directly in the trust anchor store narrows the Verifier's trust to only specific devices rather than all devices the Endorser might vouch for, such as all devices manufactured by the same manufacturer in the case that the Endorser is a manufacturer.

Such narrowing is often important since physical possession of a device can also be used to conduct a number of attacks, and so a device in a physically secure environment (such as one's own premises) may be considered trusted whereas devices owned by others would not be. This often results in a desire to either have the owner run their own Endorser that would only endorse devices one owns, or to use Attesters directly in the trust anchor store. When there are many Attesters owned, the use of an Endorser enables better scalability.

That is, a Verifier might appraise the trustworthiness of an application component, operating system component, or service under the assumption that information provided about it by the lower-layer firmware or software is true. A stronger level of assurance of security comes when information can be vouched for by hardware or by ROM code, especially if such hardware is physically resistant to hardware tampering. In most cases, components that have to be vouched for via Endorsements because no Evidence is generated about them are referred to as roots of trust.

The manufacturer having arranged for an Attesting Environment to be provisioned with key material with which to sign Evidence, the Verifier is then provided with some way of verifying the signature on the Evidence. This may be in the form of an appropriate trust anchor, or the Verifier may be provided with a database of public keys (rather than certificates) or even carefully curated and secured lists of symmetric keys.

The nature of how the Verifier manages to validate the signatures produced by the Attester is critical to the secure operation of a remote attestation system, but is not the subject of standardization within this architecture.

A conveyance protocol that provides authentication and integrity protection can be used to convey Evidence that is otherwise unprotected (e.g., not signed). Appropriate conveyance of unprotected Evidence (e.g., [I-D.birkholz-rats-uccs]) relies on the following conveyance protocol's protection capabilities:

1. The key material used to authenticate and integrity protect the conveyance channel is trusted by the Verifier to speak for the Attesting Environment(s) that collected Claims about the Target Environment(s).
2. All unprotected Evidence that is conveyed is supplied exclusively by the Attesting Environment that has the key material that protects the conveyance channel
3. A trusted environment protects the conveyance channel's key material which may depend on other Attesting Environments with equivalent strength protections.

As illustrated in [I-D.birkholz-rats-uccs], an entity that receives unprotected Evidence via a trusted conveyance channel always takes on the responsibility of vouching for the Evidence's authenticity and freshness. If protected Evidence is generated, the Attester's Attesting Environments take on that responsibility. In cases where unprotected Evidence is processed by a Verifier, Relying Parties have

to trust that the Verifier is capable of handling Evidence in a manner that preserves the Evidence's authenticity and freshness. Generating and conveying unprotected Evidence always creates significant risk and the benefits of that approach have to be carefully weighed against potential drawbacks.

See Section 12 for discussion on security strength.

7.5. Endorser, Reference Value Provider, and Verifier Owner

In some scenarios, the Endorser, Reference Value Provider, and Verifier Owner may need to trust the Verifier before giving the Endorsement, Reference Values, or appraisal policy to it. This can be done similarly to how a Relying Party might establish trust in a Verifier.

As discussed in Section 7.3, authentication or attestation in both directions might be needed, in which case typically one side's identity or Evidence must be considered safe to share with an untrusted entity, in order to bootstrap the sequence. See Section 11 for more discussion.

8. Conceptual Messages

Figure 1 illustrates the flow of a conceptual messages between various roles. This section provides additional elaboration and implementation considerations. It is the responsibility of protocol specifications to define the actual data format and semantics of any relevant conceptual messages.

8.1. Evidence

Evidence is a set of Claims about the target environment that reveal operational status, health, configuration or construction that have security relevance. Evidence is appraised by a Verifier to establish its relevance, compliance, and timeliness. Claims need to be collected in a manner that is reliable such that a Target Environment cannot lie to the Attesting Environment about its trustworthiness properties. Evidence needs to be securely associated with the target environment so that the Verifier cannot be tricked into accepting Claims originating from a different environment (that may be more trustworthy). Evidence also must be protected from man-in-the-middle attackers who may observe, change or misdirect Evidence as it travels from Attester to Verifier. The timeliness of Evidence can be captured using Claims that pinpoint the time or interval when changes in operational status, health, and so forth occur.

8.2. Endorsements

An Endorsement is a secure statement that some entity (e.g., a manufacturer) vouches for the integrity of the device's various capabilities such as claims collection, signing, launching code, transitioning to other environments, storing secrets, and more. For example, if the device's signing capability is in hardware, then an Endorsement might be a manufacturer certificate that signs a public key whose corresponding private key is only known inside the device's hardware. Thus, when Evidence and such an Endorsement are used together, an appraisal procedure can be conducted based on appraisal policies that may not be specific to the device instance, but merely specific to the manufacturer providing the Endorsement. For example, an appraisal policy might simply check that devices from a given manufacturer have information matching a set of Reference Values, or an appraisal policy might have a set of more complex logic on how to appraise the validity of information.

However, while an appraisal policy that treats all devices from a given manufacturer the same may be appropriate for some use cases, it would be inappropriate to use such an appraisal policy as the sole means of authorization for use cases that wish to constrain which compliant devices are considered authorized for some purpose. For example, an enterprise using remote attestation for Network Endpoint Assessment [RFC5209] may not wish to let every healthy laptop from the same manufacturer onto the network, but instead only want to let devices that it legally owns onto the network. Thus, an Endorsement may be helpful information in authenticating information about a device, but is not necessarily sufficient to authorize access to resources which may need device-specific information such as a public key for the device or component or user on the device.

8.3. Reference Values

Reference Values used in appraisal procedures come from a Reference Value Provider and are then used by the Verifier to compare to Evidence. Reference Values with matching Evidence produces acceptable Claims. Additionally, appraisal policy may play a role in determining the acceptance of Claims.

8.4. Attestation Results

Attestation Results are the input used by the Relying Party to decide the extent to which it will trust a particular Attester, and allow it to access some data or perform some operation.

Attestation Results may carry a boolean value indicating compliance or non-compliance with a Verifier's appraisal policy, or may carry a richer set of Claims about the Attester, against which the Relying Party applies its Appraisal Policy for Attestation Results.

The quality of the Attestation Results depends upon the ability of the Verifier to evaluate the Attester. Different Attesters have a different `_Strength of Function_` [strengthoffunction], which results in the Attestation Results being qualitatively different in strength.

An Attestation Result that indicates non-compliance can be used by an Attester (in the passport model) or a Relying Party (in the background-check model) to indicate that the Attester should not be treated as authorized and may be in need of remediation. In some cases, it may even indicate that the Evidence itself cannot be authenticated as being correct.

By default, the Relying Party does not believe the Attester to be compliant. Upon receipt of an authentic Attestation Result and given the Appraisal Policy for Attestation Results is satisfied, the Attester is allowed to perform the prescribed actions or access. The simplest such appraisal policy might authorize granting the Attester full access or control over the resources guarded by the Relying Party. A more complex appraisal policy might involve using the information provided in the Attestation Result to compare against expected values, or to apply complex analysis of other information contained in the Attestation Result.

Thus, Attestation Results can contain detailed information about an Attester, which can include privacy sensitive information as discussed in section Section 11. Unlike Evidence, which is often very device- and vendor-specific, Attestation Results can be vendor-neutral, if the Verifier has a way to generate vendor-agnostic information based on the appraisal of vendor-specific information in Evidence. This allows a Relying Party's appraisal policy to be simpler, potentially based on standard ways of expressing the information, while still allowing interoperability with heterogeneous devices.

Finally, whereas Evidence is signed by the device (or indirectly by a manufacturer, if Endorsements are used), Attestation Results are signed by a Verifier, allowing a Relying Party to only need a trust relationship with one entity, rather than a larger set of entities, for purposes of its appraisal policy.

8.5. Appraisal Policies

The Verifier, when appraising Evidence, or the Relying Party, when appraising Attestation Results, checks the values of matched Claims against constraints specified in its appraisal policy. Examples of such constraints checking include:

- * comparison for equality against a Reference Value, or
- * a check for being in a range bounded by Reference Values, or
- * membership in a set of Reference Values, or
- * a check against values in other Claims.

Upon completing all appraisal policy constraints, the remaining Claims are accepted as input toward determining Attestation Results, when appraising Evidence, or as input to a Relying Party, when appraising Attestation Results.

9. Claims Encoding Formats

The following diagram illustrates a relationship to which remote attestation is desired to be added:

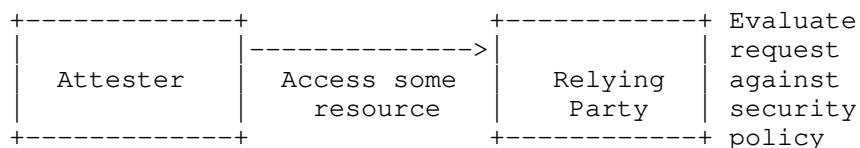


Figure 8: Typical Resource Access

In this diagram, the protocol between Attester and a Relying Party can be any new or existing protocol (e.g., HTTP(S), COAP(S), ROLIE [RFC8322], 802.1x, OPC UA [OPCUA], etc.), depending on the use case.

Typically, such protocols already have mechanisms for passing security information for authentication and authorization purposes. Common formats include JWTs [RFC7519], CWTs [RFC8392], and X.509 certificates.

Retrofitting already deployed protocols with remote attestation requires adding RATS conceptual messages to the existing data flows. This must be done in a way that does not degrade the security properties of the systems involved and should use native extension mechanisms provided by the underlying protocol. For example, if a TLS handshake is to be extended with remote attestation capabilities,

attestation Evidence may be embedded in an ad-hoc X.509 certificate extension (e.g., [TCG-DICE]), or into a new TLS Certificate Type (e.g., [I-D.tschofenig-tls-cwt]).

Especially for constrained nodes there is a desire to minimize the amount of parsing code needed in a Relying Party, in order to both minimize footprint and to minimize the attack surface. While it would be possible to embed a CWT inside a JWT, or a JWT inside an X.509 extension, etc., there is a desire to encode the information natively in a format that is already supported by the Relying Party.

This motivates having a common "information model" that describes the set of remote attestation related information in an encoding-agnostic way, and allowing multiple encoding formats (CWT, JWT, X.509, etc.) that encode the same information into the Claims format needed by the Relying Party.

The following diagram illustrates that Evidence and Attestation Results might be expressed via multiple potential encoding formats, so that they can be conveyed by various existing protocols. It also motivates why the Verifier might also be responsible for accepting Evidence that encodes Claims in one format, while issuing Attestation Results that encode Claims in a different format.

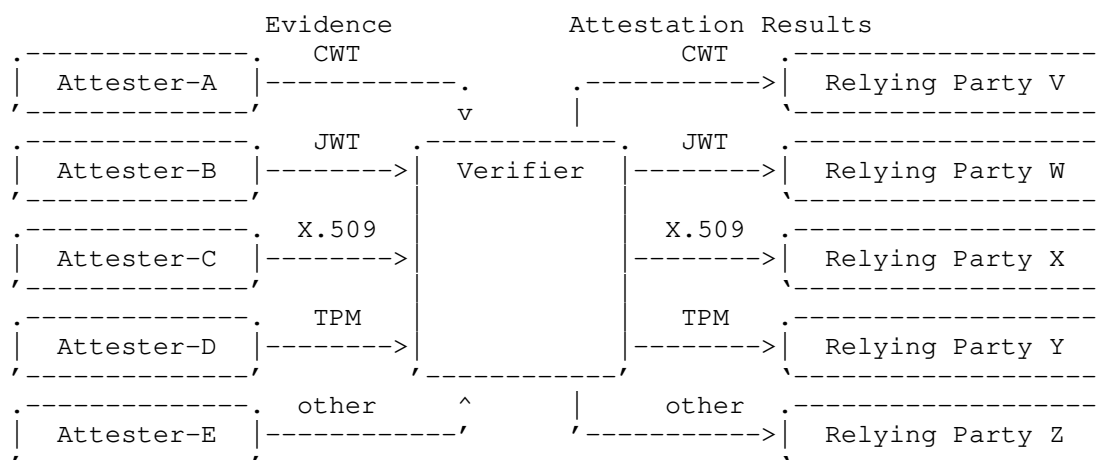


Figure 9: Multiple Attesters and Relying Parties with Different Formats

10. Freshness

A Verifier or Relying Party might need to learn the point in time (i.e., the "epoch") an Evidence or Attestation Result has been produced. This is essential in deciding whether the included Claims can be considered fresh, meaning they still reflect the latest state of the Attester, and that any Attestation Result was generated using the latest Appraisal Policy for Evidence.

This section provides a number of details. It does not however define any protocol formats, the interactions shown are abstract. This section is intended for those creating protocols and solutions to understand the options available to ensure freshness. The way in which freshness is provisioned in a protocol is an architectural decision. Provisioning of freshness has an impact on the number of needed round trips in a protocol, and therefore must be made very early in the design. Different decisions will have significant impacts on resulting interoperability, which is why this section goes into sufficient detail such that choices in freshness will be compatible across interacting protocols, such as depicted in Figure 9.

Freshness is assessed based on the Appraisal Policy for Evidence or Attestation Results that compares the estimated epoch against an "expiry" threshold defined locally to that policy. There is, however, always a race condition possible in that the state of the Attester, and the appraisal policies might change immediately after the Evidence or Attestation Result was generated. The goal is merely to narrow their recentness to something the Verifier (for Evidence) or Relying Party (for Attestation Result) is willing to accept. Some flexibility on the freshness requirement is a key component for enabling caching and reuse of both Evidence and Attestation Results, which is especially valuable in cases where their computation uses a substantial part of the resource budget (e.g., energy in constrained devices).

There are three common approaches for determining the epoch of Evidence or an Attestation Result.

10.1. Explicit Timekeeping using Synchronized Clocks

The first approach is to rely on synchronized and trustworthy clocks, and include a signed timestamp (see [I-D.birkholz-rats-tuda]) along with the Claims in the Evidence or Attestation Result. Timestamps can also be added on a per-Claim basis to distinguish the time of generation of Evidence or Attestation Result from the time that a specific Claim was generated. The clock's trustworthiness can generally be established via Endorsements and typically requires

additional Claims about the signer's time synchronization mechanism.

In some use cases, however, a trustworthy clock might not be available. For example, in many Trusted Execution Environments (TEEs) today, a clock is only available outside the TEE and so cannot be trusted by the TEE.

10.2. Implicit Timekeeping using Nonces

A second approach places the onus of timekeeping solely on the Verifier (for Evidence) or the Relying Party (for Attestation Results), and might be suitable, for example, in case the Attester does not have a trustworthy clock or time synchronization is otherwise impaired. In this approach, a non-predictable nonce is sent by the appraising entity, and the nonce is then signed and included along with the Claims in the Evidence or Attestation Result. After checking that the sent and received nonces are the same, the appraising entity knows that the Claims were signed after the nonce was generated. This allows associating a "rough" epoch to the Evidence or Attestation Result. In this case the epoch is said to be rough because:

- * The epoch applies to the entire Claim set instead of a more granular association, and
- * The time between the creation of Claims and the collection of Claims is indistinguishable.

10.3. Implicit Timekeeping using Epoch IDs

A third approach relies on having epoch identifiers (or "IDs") periodically sent to both the sender and receiver of Evidence or Attestation Results by some "Epoch ID Distributor".

Epoch IDs are different from nonces as they can be used more than once and can even be used by more than one entity at the same time. Epoch IDs are different from timestamps as they do not have to convey information about a point in time, i.e., they are not necessarily monotonically increasing integers.

Like the nonce approach, this allows associating a "rough" epoch without requiring a trustworthy clock or time synchronization in order to generate or appraise the freshness of Evidence or Attestation Results. Only the Epoch ID Distributor requires access to a clock so it can periodically send new epoch IDs.

The most recent epoch ID is included in the produced Evidence or Attestation Results, and the appraising entity can compare the epoch ID in received Evidence or Attestation Results against the latest epoch ID it received from the Epoch ID Distributor to determine if it is within the current epoch. An actual solution also needs to take into account race conditions when transitioning to a new epoch, such as by using a counter signed by the Epoch ID Distributor as the epoch ID, or by including both the current and previous epoch IDs in messages and/or checks, by requiring retries in case of mismatching epoch IDs, or by buffering incoming messages that might be associated with a epoch ID that the receiver has not yet obtained.

More generally, in order to prevent an appraising entity from generating false negatives (e.g., discarding Evidence that is deemed stale even if it is not), the appraising entity should keep an "epoch window" consisting of the most recently received epoch IDs. The depth of such epoch window is directly proportional to the maximum network propagation delay between the first to receive the epoch ID and the last to receive the epoch ID, and it is inversely proportional to the epoch duration. The appraising entity shall compare the epoch ID carried in the received Evidence or Attestation Result with the epoch IDs in its epoch window to find a suitable match.

Whereas the nonce approach typically requires the appraising entity to keep state for each nonce generated, the epoch ID approach minimizes the state kept to be independent of the number of Attesters or Verifiers from which it expects to receive Evidence or Attestation Results, as long as all use the same Epoch ID Distributor.

10.4. Discussion

Implicit and explicit timekeeping can be combined into hybrid mechanisms. For example, if clocks exist and are considered trustworthy but are not synchronized, a nonce-based exchange may be used to determine the (relative) time offset between the involved peers, followed by any number of timestamp based exchanges.

It is important to note that the actual values in Claims might have been generated long before the Claims are signed. If so, it is the signer's responsibility to ensure that the values are still correct when they are signed. For example, values generated at boot time might have been saved to secure storage until network connectivity is established to the remote Verifier and a nonce is obtained.

A more detailed discussion with examples appears in Section 16.

For a discussion on the security of epoch IDs see Section 12.3.

11. Privacy Considerations

The conveyance of Evidence and the resulting Attestation Results reveal a great deal of information about the internal state of a device as well as potentially any users of the device.

In many cases, the whole point of attestation procedures is to provide reliable information about the type of the device and the firmware/software that the device is running.

This information might be particularly interesting to many attackers. For example, knowing that a device is running a weak version of firmware provides a way to aim attacks better.

In some circumstances, if an attacker can become aware of Endorsements, Reference Values, or appraisal policies, it could potentially provide an attacker with insight into defensive mitigations. It is recommended that attention be paid to confidentiality of such information.

Additionally, many Claims in Evidence, many Claims in Attestation Results, and appraisal policies potentially contain Personally Identifying Information (PII) depending on the end-to-end use case of the remote attestation procedure. Remote attestation that includes containers and applications, e.g., a blood pressure monitor, may further reveal details about specific systems or users.

In some cases, an attacker may be able to make inferences about the contents of Evidence from the resulting effects or timing of the processing. For example, an attacker might be able to infer the value of specific Claims if it knew that only certain values were accepted by the Relying Party.

Conceptual messages (see Section 8) carrying sensitive or confidential information are expected to be integrity protected (i.e., either via signing or a secure channel) and optionally might be confidentiality protected via encryption. If there isn't confidentiality protection of conceptual messages themselves, the underlying conveyance protocol should provide these protections.

As Evidence might contain sensitive or confidential information, Attesters are responsible for only sending such Evidence to trusted Verifiers. Some Attesters might want a stronger level of assurance of the trustworthiness of a Verifier before sending Evidence to it. In such cases, an Attester can first act as a Relying Party and ask for the Verifier's own Attestation Result, and appraising it just as a Relying Party would appraise an Attestation Result for any other purpose.

Another approach to deal with Evidence is to remove PII from the Evidence while still being able to verify that the Attester is one of a large set. This approach is often called "Direct Anonymous Attestation". See [CCC-DeepDive] section 6.2 and [I-D.ietf-rats-daa] for more discussion.

12. Security Considerations

This document provides an architecture for doing remote attestation. No specific wire protocol is documented here. Without a specific proposal to compare against, it is impossible to know if the security threats listed below have been mitigated well.

The security considerations below should be read as being essentially requirements against realizations of the RATS Architecture. Some threats apply to protocols, some are against implementations (code), and some threats are against physical infrastructure (such as factories).

The fundamental purpose of the RATS architecture is to allow a Relying Party to establish a basis for trusting the Attester.

12.1. Attester and Attestation Key Protection

Implementers need to pay close attention to the protection of the Attester and the manufacturing processes for provisioning attestation key material. If either of these are compromised, intended levels of assurance for RATS are compromised because attackers can forge Evidence or manipulate the Attesting Environment. For example, a Target Environment should not be able to tamper with the Attesting Environment that measures it, by isolating the two environments from each other in some way.

Remote attestation applies to use cases with a range of security requirements, so the protections discussed here range from low to high security where low security may be limited to application or process isolation by the device's operating system, and high security may involve specialized hardware to defend against physical attacks on a chip.

12.1.1. On-Device Attester and Key Protection

It is assumed that an Attesting Environment is sufficiently isolated from the Target Environment it collects Claims about and that it signs the resulting Claims set with an attestation key, so that the Target Environment cannot forge Evidence about itself. Such an isolated environment might be provided by a process, a dedicated chip, a TEE, a virtual machine, or another secure mode of operation.

The Attesting Environment must be protected from unauthorized modification to ensure it behaves correctly. Confidentiality protection of the Attesting Environment's signing key is vital so it cannot be misused to forge Evidence.

In many cases the user or owner of a device that includes the role of Attester must not be able to modify or extract keys from the Attesting Environments, to prevent creating forged Evidence. Some common examples include the user of a mobile phone or FIDO authenticator.

Measures for a minimally protected system might include process or application isolation provided by a high-level operating system, and restricted access to root or system privileges. In contrast, For really simple single-use devices that don't use a protected mode operating system, like a Bluetooth speaker, the only factual isolation might be the sturdy housing of the device.

Measures for a moderately protected system could include a special restricted operating environment, such as a TEE. In this case, only security-oriented software has access to the Attester and key material.

Measures for a highly protected system could include specialized hardware that is used to provide protection against chip decapping attacks, power supply and clock glitching, faulting injection and RF and power side channel attacks.

12.1.2. Attestation Key Provisioning Processes

Attestation key provisioning is the process that occurs in the factory or elsewhere to establish signing key material on the device and the validation key material off the device. Sometimes this procedure is referred to as personalization or customization.

The keys generated in the factory, whether generated in the device or off-device by the factory SHOULD be generated by a Cryptographically Strong Sequence ([RFC4086], Section 6.2).

12.1.2.1. Off-Device Key Generation

One way to provision key material is to first generate it external to the device and then copy the key onto the device. In this case, confidentiality protection of the generator, as well as for the path over which the key is provisioned, is necessary. The manufacturer needs to take care to protect corresponding key material with measures appropriate for its value.

The degree of protection afforded to this key material can vary by the intended function of the device and the specific practices of the device manufacturer or integrator. The confidentiality protection is fundamentally based upon some amount of physical protection: while encryption is often used to provide confidentiality when a key is conveyed across a factory, where the attestation key is created or applied, it must be available in an unencrypted form. The physical protection can therefore vary from situations where the key is unencrypted only within carefully controlled secure enclaves within silicon, to situations where an entire facility is considered secure, by the simple means of locked doors and limited access.

The cryptography that is used to enable confidentiality protection of the attestation key comes with its own requirements to be secured. This results in recursive problems, as the key material used to provision attestation keys must again somehow have been provisioned securely beforehand (requiring an additional level of protection, and so on).

So, this is why, in general, a combination of some physical security measures and some cryptographic measures is used to establish confidentiality protection.

12.1.2.2. On-Device Key Generation

When key material is generated within a device and the secret part of it never leaves the device, then the problem may lessen. For public-key cryptography, it is, by definition, not necessary to maintain confidentiality of the public key: however integrity of the chain of custody of the public key is necessary in order to avoid attacks where an attacker is able get a key they control endorsed.

To summarize: attestation key provisioning must ensure that only valid attestation key material is established in Attesters.

12.2. Integrity Protection

Any solution that conveys information in any conceptual message (see Section 8) must support end-to-end integrity protection and replay attack prevention, and often also needs to support additional security properties, including:

- * end-to-end encryption,
- * denial of service protection,
- * authentication,

- * auditing,
- * fine grained access controls, and
- * logging.

Section 10 discusses ways in which freshness can be used in this architecture to protect against replay attacks.

To assess the security provided by a particular appraisal policy, it is important to understand the strength of the root of trust, e.g., whether it is mutable software, or firmware that is read-only after boot, or immutable hardware/ROM.

It is also important that the appraisal policy was itself obtained securely. If an attacker can configure or modify appraisal policies, Endorsements or Reference Values for a Relying Party or for a Verifier, then integrity of the process is compromised.

Security protections in RATS may be applied at different layers, whether by a conveyance protocol, or an information encoding format. This architecture expects conceptual messages to be end-to-end protected based on the role interaction context. For example, if an Attester produces Evidence that is relayed through some other entity that doesn't implement the Attester or the intended Verifier roles, then the relaying entity should not expect to have access to the Evidence.

12.3. Epoch ID-based Attestation

Epoch IDs, described in Section 10.3, can be tampered with, replayed, dropped, delayed, and reordered by an attacker.

An attacker could be either external or belong to the distribution group, for example, if one of the Attester entities have been compromised.

An attacker who is able to tamper with epoch IDs can potentially lock all the participants in a certain epoch of choice for ever, effectively freezing time. This is problematic since it destroys the ability to ascertain freshness of Evidence and Attestation Results.

To mitigate this threat, the transport should be at least integrity protected and provide origin authentication.

Selective dropping of epoch IDs is equivalent to pinning the victim node to a past epoch. An attacker could drop epoch IDs to only some entities and not others, which will typically result in a denial of service due to the permanent staleness of the Attestation Result or Evidence.

Delaying or reordering epoch IDs is equivalent to manipulating the victim's timeline at will. This ability could be used by a malicious actor (e.g., a compromised router) to mount a confusion attack where, for example, a Verifier is tricked into accepting Evidence coming from a past epoch as fresh, while in the meantime the Attester has been compromised.

Reordering and dropping attacks are mitigated if the transport provides the ability to detect reordering and drop. However, the delay attack described above can't be thwarted in this manner.

12.4. Trust Anchor Protection

As noted in Section 7, Verifiers and Relying Parties have trust anchor stores that must be secured. [RFC6024] contains more discussion of trust anchor store requirements for protecting public keys. Section 6 of [NIST-800-57-p1] contains a comprehensive treatment of the topic, including the protection of symmetric key material. Specifically, a trust anchor store must resist modification against unauthorized insertion, deletion, and modification. Additionally, if the trust anchor is a symmetric key, the trust anchor store must not allow unauthorized read.

If certificates are used as trust anchors, Verifiers and Relying Parties are also responsible for validating the entire certificate path up to the trust anchor, which includes checking for certificate revocation. See Section 6 of [RFC5280] for details.

13. IANA Considerations

This document does not require any actions by IANA.

14. Acknowledgments

Special thanks go to Joerg Borchert, Nancy Cam-Winget, Jessica Fitzgerald-McKay, Diego Lopez, Laurence Lundblade, Paul Rowe, Hannes Tschofenig, Frank Xia, and David Wooten.

15. Notable Contributions

Thomas Hardjono created initial versions of the terminology section in collaboration with Ned Smith. Eric Voit provided the conceptual separation between Attestation Provision Flows and Attestation Evidence Flows. Monty Wisemen created the content structure of the first three architecture drafts. Carsten Bormann provided many of the motivational building blocks with respect to the Internet Threat Model.

16. Appendix A: Time Considerations

Section 10 discussed various issues and requirements around freshness of evidence, and summarized three approaches that might be used by different solutions to address them. This appendix provides more details with examples to help illustrate potential approaches, to inform those creating specific solutions.

The table below defines a number of relevant events, with an ID that is used in subsequent diagrams. The times of said events might be defined in terms of an absolute clock time, such as the Coordinated Universal Time timescale, or might be defined relative to some other timestamp or timeticks counter, such as a clock resetting its epoch each time it is powered on.

ID	Event	Explanation of event
VG	Value generated	A value to appear in a Claim was created. In some cases, a value may have technically existed before an Attester became aware of it but the Attester might have no idea how long it has had that value. In such a case, the Value created time is the time at which the Claim containing the copy of the value was created.
NS	Nonce sent	A nonce not predictable to an Attester (recentness & uniqueness) is sent to an Attester.
NR	Nonce relayed	A nonce is relayed to an Attester by another entity.
IR	Epoch ID received	An epoch ID is successfully received and processed by an entity.
EG	Evidence	An Attester creates Evidence from collected

	generation	Claims.
ER	Evidence relayed	A Relying Party relays Evidence to a Verifier.
RG	Result generation	A Verifier appraises Evidence and generates an Attestation Result.
RR	Result relayed	A Relying Party relays an Attestation Result to a Relying Party.
RA	Result appraised	The Relying Party appraises Attestation Results.
OP	Operation performed	The Relying Party performs some operation requested by the Attester via a resource access protocol as depicted in Figure 8, e.g., across a session created earlier at time(RA).
RX	Result expiry	An Attestation Result should no longer be accepted, according to the Verifier that generated it.

Table 1

Using the table above, a number of hypothetical examples of how a solution might be built are illustrated below. This list is not intended to be complete, but is just representative enough to highlight various timing considerations.

All times are relative to the local clocks, indicated by an "_a" (Attester), "_v" (Verifier), or "_r" (Relying Party) suffix.

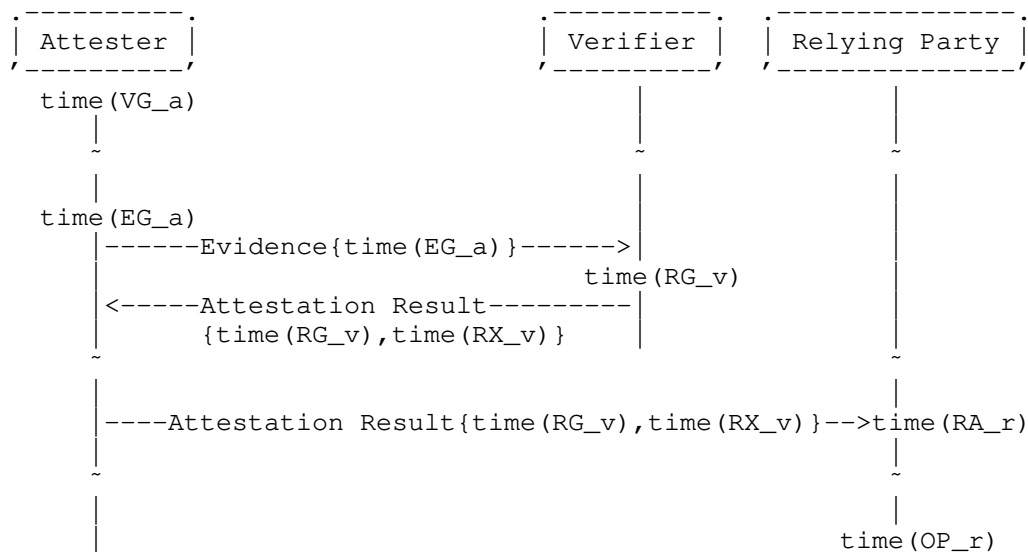
Times with an appended Prime (') indicate a second instance of the same event.

How and if clocks are synchronized depends upon the model.

In the figures below, curly braces indicate containment. For example, the notation Evidence{foo} indicates that 'foo' is contained in the Evidence and is thus covered by its signature.

16.1. Example 1: Timestamp-based Passport Model Example

The following example illustrates a hypothetical Passport Model solution that uses timestamps and requires roughly synchronized clocks between the Attester, Verifier, and Relying Party, which depends on using a secure clock synchronization mechanism. As a result, the receiver of a conceptual message containing a timestamp can directly compare it to its own clock and timestamps.



The Verifier can check whether the Evidence is fresh when appraising it at $\text{time}(\text{RG}_v)$ by checking $\text{time}(\text{RG}_v) - \text{time}(\text{EG}_a) < \text{Threshold}$, where the Verifier's threshold is large enough to account for the maximum permitted clock skew between the Verifier and the Attester.

If $\text{time}(\text{VG}_a)$ is also included in the Evidence along with the Claim value generated at that time, and the Verifier decides that it can trust the $\text{time}(\text{VG}_a)$ value, the Verifier can also determine whether the Claim value is recent by checking $\text{time}(\text{RG}_v) - \text{time}(\text{VG}_a) < \text{Threshold}$. The threshold is decided by the Appraisal Policy for Evidence, and again needs to take into account the maximum permitted clock skew between the Verifier and the Attester.

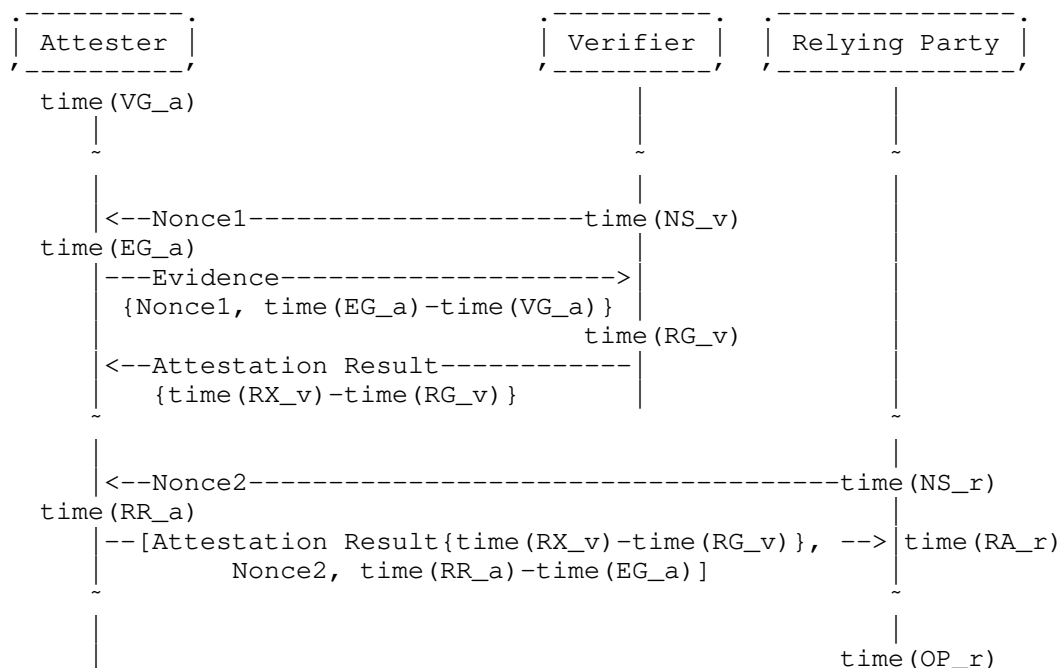
The Relying Party can check whether the Attestation Result is fresh when appraising it at $\text{time}(\text{RA}_r)$ by checking $\text{time}(\text{RA}_r) - \text{time}(\text{RG}_v) < \text{Threshold}$, where the Relying Party's threshold is large enough to account for the maximum permitted clock skew between the Relying Party and the Verifier. The result might then be used for some time (e.g., throughout the lifetime of a connection established at

$\text{time}(\text{RA}_r)$). The Relying Party must be careful, however, to not allow continued use beyond the period for which it deems the Attestation Result to remain fresh enough. Thus, it might allow use (at $\text{time}(\text{OP}_r)$) as long as $\text{time}(\text{OP}_r) - \text{time}(\text{RG}_v) < \text{Threshold}$. However, if the Attestation Result contains an expiry time $\text{time}(\text{RX}_v)$ then it could explicitly check $\text{time}(\text{OP}_r) < \text{time}(\text{RX}_v)$.

16.2. Example 2: Nonce-based Passport Model Example

The following example illustrates a hypothetical Passport Model solution that uses nonces instead of timestamps. Compared to the timestamp-based example, it requires an extra round trip to retrieve a nonce, and requires that the Verifier and Relying Party track state to remember the nonce for some period of time.

The advantage is that it does not require that any clocks are synchronized. As a result, the receiver of a conceptual message containing a timestamp cannot directly compare it to its own clock or timestamps. Thus we use a suffix ("a" for Attester, "v" for Verifier, and "r" for Relying Party) on the IDs below indicating which clock generated them, since times from different clocks cannot be compared. Only the delta between two events from the sender can be used by the receiver.



In this example solution, the Verifier can check whether the Evidence is fresh at time(RG_v) by verifying that $\text{time(RG_v)} - \text{time(NS_v)} < \text{Threshold}$.

The Verifier cannot, however, simply rely on a Nonce to determine whether the value of a Claim is recent, since the Claim value might have been generated long before the nonce was sent by the Verifier. However, if the Verifier decides that the Attester can be trusted to correctly provide the delta time $\text{time(EG_a)} - \text{time(VG_a)}$, then it can determine recency by checking $\text{time(RG_v)} - \text{time(NS_v)} + \text{time(EG_a)} - \text{time(VG_a)} < \text{Threshold}$.

Similarly if, based on an Attestation Result from a Verifier it trusts, the Relying Party decides that the Attester can be trusted to correctly provide time deltas, then it can determine whether the Attestation Result is fresh by checking $\text{time(OP_r)} - \text{time(NS_r)} + \text{time(RR_a)} - \text{time(EG_a)} < \text{Threshold}$. Although the Nonce2 and $\text{time(RR_a)} - \text{time(EG_a)}$ values cannot be inside the Attestation Result, they might be signed by the Attester such that the Attestation Result vouches for the Attester's signing capability.

The Relying Party must still be careful, however, to not allow continued use beyond the period for which it deems the Attestation Result to remain valid. Thus, if the Attestation Result sends a validity lifetime in terms of $\text{time(RX_v)} - \text{time(RG_v)}$, then the Relying Party can check $\text{time(OP_r)} - \text{time(NS_r)} < \text{time(RX_v)} - \text{time(RG_v)}$.

16.3. Example 3: Epoch ID-based Passport Model Example

The example in Figure 10 illustrates a hypothetical Passport Model solution that uses epoch IDs instead of nonces or timestamps.

The Epoch ID Distributor broadcasts epoch ID I which starts a new epoch E for a protocol participant upon reception at time(IR).

The Attester generates Evidence incorporating epoch ID I and conveys it to the Verifier.

The Verifier appraises that the received epoch ID I is "fresh" according to the definition provided in Section 10.3 whereby retries are required in the case of mismatching epoch IDs, and generates an Attestation Result. The Attestation Result is conveyed to the Attester.

After the transmission of epoch ID I' a new epoch E' is established when I' is received by each protocol participant. The Attester relays the Attestation Result obtained during epoch E (associated with epoch ID I) to the Relying Party using the epoch ID for the

current epoch I' . If the Relying Party had not yet received I' , then the Attestation Result would be rejected, but in this example, it is received.

In the illustrated scenario, the epoch ID for relaying an Attestation Result to the Relying Party is current, while a previous epoch ID was used to generate Verifier evaluated evidence. This indicates that at least one epoch transition has occurred, and the Attestation Results may only be as fresh as the previous epoch. If the Relying Party remembers the previous epoch ID I during an epoch window as discussed in Section 10.3, and the message is received during that window, the Attestation Result is accepted as fresh, and otherwise it is rejected as stale.

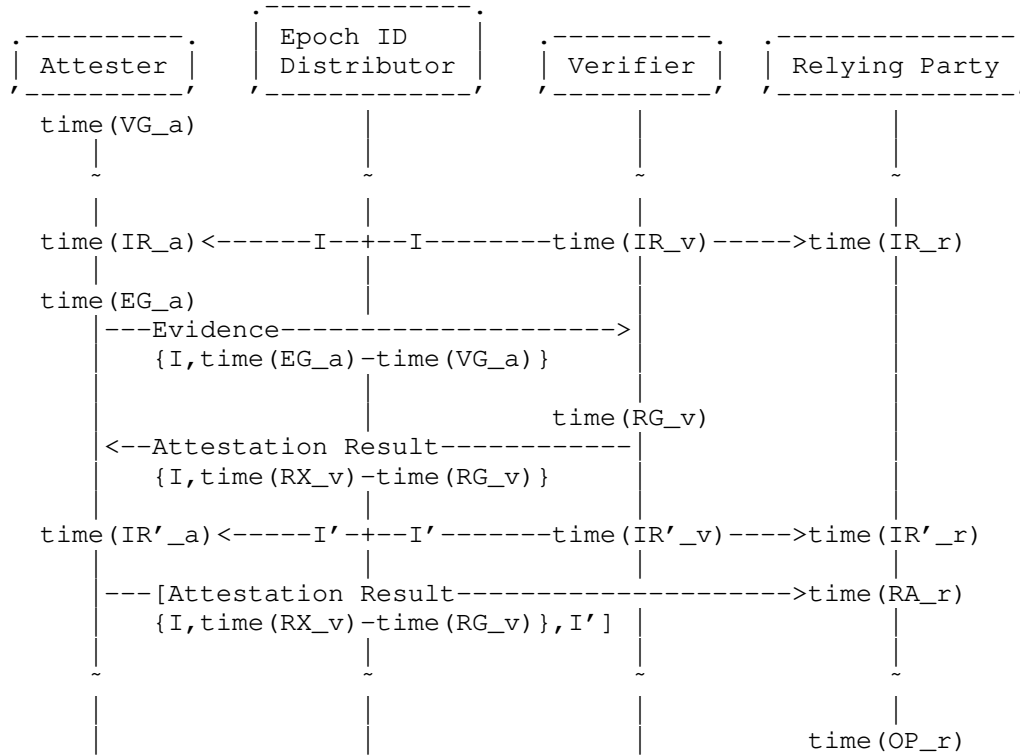
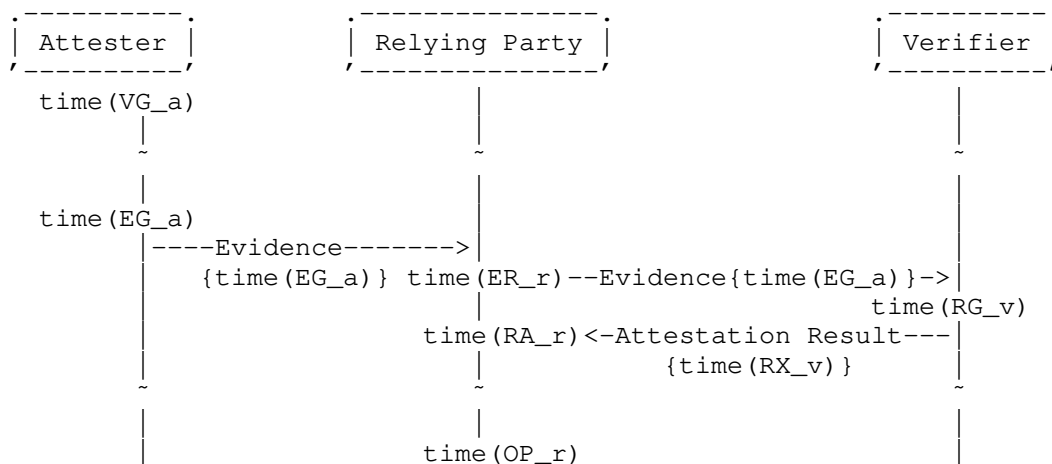


Figure 10: Epoch ID-based Passport Model

16.4. Example 4: Timestamp-based Background-Check Model Example

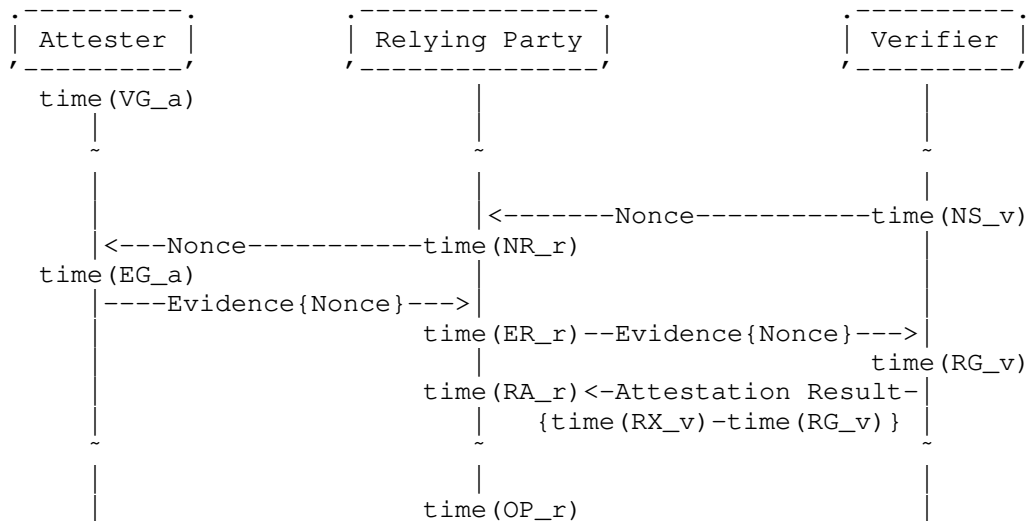
The following example illustrates a hypothetical Background-Check Model solution that uses timestamps and requires roughly synchronized clocks between the Attester, Verifier, and Relying Party.



The time considerations in this example are equivalent to those discussed under Example 1 above.

16.5. Example 5: Nonce-based Background-Check Model Example

The following example illustrates a hypothetical Background-Check Model solution that uses nonces and thus does not require that any clocks are synchronized. In this example solution, a nonce is generated by a Verifier at the request of a Relying Party, when the Relying Party needs to send one to an Attester.



The Verifier can check whether the Evidence is fresh, and whether a Claim value is recent, the same as in Example 2 above.

However, unlike in Example 2, the Relying Party can use the Nonce to determine whether the Attestation Result is fresh, by verifying that $\text{time}(\text{OP_r}) - \text{time}(\text{NR_r}) < \text{Threshold}$.

The Relying Party must still be careful, however, to not allow continued use beyond the period for which it deems the Attestation Result to remain valid. Thus, if the Attestation Result sends a validity lifetime in terms of $\text{time}(\text{RX_v}) - \text{time}(\text{RG_v})$, then the Relying Party can check $\text{time}(\text{OP_r}) - \text{time}(\text{ER_r}) < \text{time}(\text{RX_v}) - \text{time}(\text{RG_v})$.

17. References

17.1. Normative References

- [RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, DOI 10.17487/RFC5280, May 2008, <<https://www.rfc-editor.org/info/rfc5280>>.
- [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/info/rfc7519>>.
- [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/info/rfc8392>>.

17.2. Informative References

- [CCC-DeepDive] Confidential Computing Consortium, "Confidential Computing Deep Dive", n.d., <<https://confidentialcomputing.io/whitepaper-02-latest>>.
- [CTAP] FIDO Alliance, "Client to Authenticator Protocol", n.d., <<https://fidoalliance.org/specs/fido-v2.0-id-20180227/fido-client-to-authenticator-protocol-v2.0-id-20180227.html>>.

[I-D.birkholz-rats-tuda]

Fuchs, A., Birkholz, H., McDonald, I. E., and C. Bormann, "Time-Based Uni-Directional Attestation", Work in Progress, Internet-Draft, draft-birkholz-rats-tuda-06, 12 January 2022, <<https://www.ietf.org/archive/id/draft-birkholz-rats-tuda-06.txt>>.

[I-D.birkholz-rats-uccs]

Birkholz, H., O'Donoghue, J., Cam-Winget, N., and C. Bormann, "A CBOR Tag for Unprotected CWT Claims Sets", Work in Progress, Internet-Draft, draft-birkholz-rats-uccs-03, 8 March 2021, <<https://www.ietf.org/archive/id/draft-birkholz-rats-uccs-03.txt>>.

[I-D.ietf-rats-daa]

Birkholz, H., Newton, C., Chen, L., and D. Thaler, "Direct Anonymous Attestation for the Remote Attestation Procedures Architecture", Work in Progress, Internet-Draft, draft-ietf-rats-daa-00, 2 December 2021, <<https://www.ietf.org/archive/id/draft-ietf-rats-daa-00.txt>>.

[I-D.ietf-teep-architecture]

Pei, M., Tschofenig, H., Thaler, D., and D. Wheeler, "Trusted Execution Environment Provisioning (TEEP) Architecture", Work in Progress, Internet-Draft, draft-ietf-teep-architecture-15, 12 July 2021, <<https://www.ietf.org/archive/id/draft-ietf-teep-architecture-15.txt>>.

[I-D.tschofenig-rats-psa-token]

Tschofenig, H., Frost, S., Brossard, M., Shaw, A., and T. Fossati, "Arm's Platform Security Architecture (PSA) Attestation Token", Work in Progress, Internet-Draft, draft-tschofenig-rats-psa-token-08, 24 March 2021, <<https://www.ietf.org/archive/id/draft-tschofenig-rats-psa-token-08.txt>>.

[I-D.tschofenig-tls-cwt]

Tschofenig, H. and M. Brossard, "Using CBOR Web Tokens (CWTs) in Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)", Work in Progress, Internet-Draft, draft-tschofenig-tls-cwt-02, 13 July 2020, <<https://www.ietf.org/archive/id/draft-tschofenig-tls-cwt-02.txt>>.

- [NIST-800-57-p1] Barker, E., "Recommendation for Key Management: Part 1 - General", May 2020, <<https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-57pt1r5.pdf>>.
- [OPCUA] OPC Foundation, "OPC Unified Architecture Specification, Part 2: Security Model, Release 1.03", OPC 10000-2 , 25 November 2015, <<https://opcfoundation.org/developer-tools/specifications-unified-architecture/part-2-security-model/>>.
- [RFC4086] Eastlake 3rd, D., Schiller, J., and S. Crocker, "Randomness Requirements for Security", BCP 106, RFC 4086, DOI 10.17487/RFC4086, June 2005, <<https://www.rfc-editor.org/info/rfc4086>>.
- [RFC4949] Shirey, R., "Internet Security Glossary, Version 2", FYI 36, RFC 4949, DOI 10.17487/RFC4949, August 2007, <<https://www.rfc-editor.org/info/rfc4949>>.
- [RFC5209] Sangster, P., Khosravi, H., Mani, M., Narayan, K., and J. Tardo, "Network Endpoint Assessment (NEA): Overview and Requirements", RFC 5209, DOI 10.17487/RFC5209, June 2008, <<https://www.rfc-editor.org/info/rfc5209>>.
- [RFC6024] Reddy, R. and C. Wallace, "Trust Anchor Management Requirements", RFC 6024, DOI 10.17487/RFC6024, October 2010, <<https://www.rfc-editor.org/info/rfc6024>>.
- [RFC8322] Field, J., Banghart, S., and D. Waltermire, "Resource-Oriented Lightweight Information Exchange (ROLIE)", RFC 8322, DOI 10.17487/RFC8322, February 2018, <<https://www.rfc-editor.org/info/rfc8322>>.
- [strengthoffunction] NISC, "Strength of Function", n.d., <https://csrc.nist.gov/glossary/term/strength_of_function>.
- [TCG-DICE] Trusted Computing Group, "DICE Certificate Profiles", n.d., <https://trustedcomputinggroup.org/wp-content/uploads/DICE-Certificate-Profiles-r01_3june2020-1.pdf>.

[TCG-DICE-SIBDA]

Trusted Computing Group, "Symmetric Identity Based Device Attestation for DICE", 24 July 2019, <https://trustedcomputinggroup.org/wp-content/uploads/TCG_DICE_SymIDAttest_v1_r0p94_pubrev.pdf>.

[TCGarch] Trusted Computing Group, "Trusted Platform Module Library - Part 1: Architecture", 8 November 2019, <https://trustedcomputinggroup.org/wp-content/uploads/TCG_TPM2_r1p59_Part1_Architecture_pub.pdf>.

[WebAuthN] W3C, "Web Authentication: An API for accessing Public Key Credentials", n.d., <<https://www.w3.org/TR/webauthn-1/>>.

Contributors

Monty Wiseman

Email: montywiseman32@gmail.com

Liang Xia

Email: frank.xialiang@huawei.com

Laurence Lundblade

Email: lg1@island-resort.com

Eliot Lear

Email: ellear@cisco.com

Jessica Fitzgerald-McKay

Sarah C. Helbe

Andrew Guinn

Peter Loscocco

Email: pete.loscocco@gmail.com

Eric Voit

Thomas Fossati

Email: thomas.fossati@arm.com

Paul Rowe

Carsten Bormann

Email: cabo@tzi.org

Giri Mandyam

Email: mandyam@qti.qualcomm.com

Kathleen Moriarty

Email: kathleen.moriarty.ietf@gmail.com

Guy Fedorkow

Email: gfedorkow@juniper.net

Simon Frost

Email: Simon.Frost@arm.com

Authors' Addresses

Henk Birkholz
Fraunhofer SIT
Rheinstrasse 75
64295 Darmstadt
Germany

Email: henk.birkholz@sit.fraunhofer.de

Dave Thaler
Microsoft
United States of America

Email: dthaler@microsoft.com

Michael Richardson
Sandelman Software Works
Canada

Email: mcr+ietf@sandelman.ca

Ned Smith
Intel Corporation
United States of America

Email: ned.smith@intel.com

Wei Pan
Huawei Technologies

Email: william.panwei@huawei.com

RATS Working Group
Internet-Draft
Intended status: Standards Track
Expires: August 27, 2022

L. Lundblade
Security Theory LLC
G. Mandyam
J. O'Donoghue
Qualcomm Technologies Inc.
February 23, 2022

The Entity Attestation Token (EAT)
draft-ietf-rats-eat-12

Abstract

An Entity Attestation Token (EAT) provides an attested claims set that describes state and characteristics of an entity, a device like a phone, IoT device, network equipment or such. This claims set is used by a relying party, server or service to determine how much it wishes to trust the entity.

An EAT is either a CBOR Web Token (CWT) or JSON Web Token (JWT) with attestation-oriented claims. To a large degree, all this document does is extend CWT and JWT.

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 August 27, 2022.

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 Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	5
1.1. Entity Overview	6
1.2. CWT, JWT, UCCS, UJCS and DEB	7
1.3. CDDL, CBOR and JSON	8
1.4. Operating Model and RATS Architecture	8
1.4.1. Relationship between Attestation Evidence and Attestation Results	9
2. Terminology	10
3. The Claims	11
3.1. Token ID Claim (cti and jti)	11
3.2. Timestamp claim (iat)	11
3.3. Nonce Claim (nonce)	12
3.4. Universal Entity ID Claim (ueid)	12
3.5. Semi-permanent UEIDs (SUEIDs)	15
3.6. Hardware OEM Identification (oemid)	16
3.6.1. Random Number Based OEMID	16
3.6.2. IEEE Based OEMID	16
3.6.3. IANA Private Enterprise Number Based OEMID	17
3.7. Hardware Model Claim (hardware-model)	17
3.8. Hardware Version Claims (hardware-version-claims)	18
3.9. Software Name Claim	19
3.10. Software Version Claim	19
3.11. The Security Level Claim (security-level)	19
3.12. Secure Boot Claim (secure-boot)	21
3.13. Debug Status Claim (debug-status)	21
3.13.1. Enabled	22
3.13.2. Disabled	22
3.13.3. Disabled Since Boot	22
3.13.4. Disabled Permanently	22
3.13.5. Disabled Fully and Permanently	22
3.14. Including Keys	23
3.15. The Location Claim (location)	24
3.16. The Uptime Claim (uptime)	25
3.17. The Boot Odometer Claim (odometer)	25
3.18. The Boot Seed Claim (boot-seed)	25
3.19. The Intended Use Claim (intended-use)	26
3.20. The Profile Claim (profile)	27
3.21. The DLOA (Digital Letter or Approval) Claim (dloas)	27
3.22. The Software Manifests Claim (manifests)	28

3.23. The Software Evidence Claim (swevidence)	30
3.24. The SW Measurement Results Claim (swresults)	30
3.24.1. Scheme	31
3.24.2. Objective	31
3.24.3. Results	31
3.24.4. Objective Name	32
3.25. Submodules (submods)	34
3.25.1. Submodule Types	34
3.25.1.1. Submodule Claims-Set	34
3.25.1.2. Nested Token	35
3.25.1.3. Detached Submodule Digest	37
3.25.2. No Inheritance	38
3.25.3. Security Levels	38
3.25.4. Submodule Names	39
3.25.5. CDDL for submods	39
4. Unprotected JWT Claims-Sets	39
5. Detached EAT Bundles	39
6. Endorsements and Verification Keys	40
6.1. Identification Methods	41
6.1.1. COSE/JWS Key ID	41
6.1.2. JWS and COSE X.509 Header Parameters	42
6.1.3. CBOR Certificate COSE Header Parameters	42
6.1.4. Claim-Based Key Identification	42
6.2. Other Considerations	42
7. Profiles	43
7.1. Format of a Profile Document	43
7.2. List of Profile Issues	43
7.2.1. Use of JSON, CBOR or both	43
7.2.2. CBOR Map and Array Encoding	44
7.2.3. CBOR String Encoding	44
7.2.4. CBOR Preferred Serialization	44
7.2.5. COSE/JOSE Protection	44
7.2.6. COSE/JOSE Algorithms	45
7.2.7. DEB Support	45
7.2.8. Verification Key Identification	45
7.2.9. Endorsement Identification	45
7.2.10. Freshness	45
7.2.11. Required Claims	45
7.2.12. Prohibited Claims	45
7.2.13. Additional Claims	46
7.2.14. Refined Claim Definition	46
7.2.15. CBOR Tags	46
7.2.16. Manifests and Software Evidence Claims	46
8. Encoding and Collected CDDL	46
8.1. Claims-Set and CDDL for CWT and JWT	46
8.2. Encoding Data Types	47
8.2.1. Common Data Types	47
8.2.2. JSON Interoperability	47

8.2.3. Labels	48
8.3. CBOR Interoperability	48
8.3.1. EAT Constrained Device Serialization	48
8.4. Collected Common CDDL	49
8.5. Collected CDDL for CBOR	54
8.6. Collected CDDL for JSON	55
9. IANA Considerations	56
9.1. Reuse of CBOR and JSON Web Token (CWT and JWT) Claims Registries	56
9.2. Claim Characteristics	57
9.2.1. Interoperability and Relying Party Orientation	57
9.2.2. Operating System and Technology Neutral	57
9.2.3. Security Level Neutral	58
9.2.4. Reuse of Extant Data Formats	58
9.2.5. Proprietary Claims	58
9.3. Claims Registered by This Document	58
9.3.1. Claims for Early Assignment	59
9.3.2. To be Assigned Claims	62
9.3.3. Version Schemes Registered by this Document	65
9.3.4. UEID URN Registered by this Document	66
9.3.5. Tag for Detached EAT Bundle	66
10. Privacy Considerations	66
10.1. UEID and SUEID Privacy Considerations	67
10.2. Location Privacy Considerations	67
10.3. Replay Protection and Privacy	68
11. Security Considerations	68
11.1. Key Provisioning	68
11.1.1. Transmission of Key Material	69
11.2. Transport Security	69
11.3. Multiple EAT Consumers	69
12. References	70
12.1. Normative References	70
12.2. Informative References	73
Appendix A. Examples	76
A.1. Simple TEE Attestation	76
A.2. Submodules for Board and Device	77
A.3. EAT Produced by Attestation Hardware Block	79
A.4. Detached EAT Bundle	79
A.5. Key / Key Store Attestation	81
A.6. SW Measurements of an IoT Device	83
A.7. Attestation Results in JSON format	86
Appendix B. UEID Design Rationale	87
B.1. Collision Probability	87
B.2. No Use of UUID	89
Appendix C. EAT Relation to IEEE.802.1AR Secure Device Identity (DevID)	90
C.1. DevID Used With EAT	90
C.2. How EAT Provides an Equivalent Secure Device Identity	91

C.3. An X.509 Format EAT	91
C.4. Device Identifier Permanence	92
Appendix D. Changes from Previous Drafts	92
D.1. From draft-rats-eat-01	92
D.2. From draft-mandyam-rats-eat-00	92
D.3. From draft-ietf-rats-eat-01	92
D.4. From draft-ietf-rats-eat-02	93
D.5. From draft-ietf-rats-eat-03	93
D.6. From draft-ietf-rats-eat-04	93
D.7. From draft-ietf-rats-eat-05	94
D.8. From draft-ietf-rats-eat-06	94
D.9. From draft-ietf-rats-eat-07	94
D.10. From draft-ietf-rats-eat-08	94
D.11. From draft-ietf-rats-eat-09	94
D.12. From draft-ietf-rats-eat-10	95
D.13. From draft-ietf-rats-eat-11	96
Authors' Addresses	96

1. Introduction

EAT provides the definition of a base set of claims that can be made about an entity, a device, some software and/or some hardware. This claims set is received by a relying party who uses it to decide if and how it will interact with the remote entity. It may choose to not trust the entity and not interact with it. It may choose to trust it. It may partially trust it, for example allowing monetary transactions only up to a limit.

EAT defines the encoding of the claims set in CBOR [RFC8949] and JSON [RFC7159]. EAT is an extension to CBOR Web Token (CWT) [RFC8392] and JSON Web Token (JWT) [RFC7519].

The claims set is secured in transit with the same mechanisms used by CWT and JWT, in particular CBOR Object Signing and Encryption (COSE) [RFC8152] and JSON Object Signing and Encryption (JOSE) [RFC7515] [RFC7516]. Authenticity and integrity protection must always be provided. Privacy (encryption) may additionally be provided. The key material used to sign and encrypt is specifically created and provisioned for the purpose of attestation. It is the use of this key material that make the claims set "attested" rather than just some parameters sent to the relying party by the device.

EAT is focused on authenticating, identifying and characterizing implementations where implementations are devices, chips, hardware, software and such. This is distinct from protocols like TLS [RFC8446] that authenticate and identify servers and services. It is equally distinct from protocols like SASL [RFC4422] that authenticate and identify persons.

The notion of attestation is large, ranging over a broad variety of use cases and security levels. Here are a few examples of claims:

- o Make and model of manufactured consumer device
- o Make and model of a chip or processor, particularly for a security-oriented chip
- o Identification and measurement of the software running on a device
- o Configuration and state of a device
- o Environmental characteristics of a device like its GPS location
- o Formal certifications received

EAT also supports nesting of sets of claims and EAT tokens for use with complex composite devices.

This document uses the terminology and main operational model defined in [RATS.Architecture]. In particular, it can be used for RATS Attestation Evidence and Attestation Results.

1.1. Entity Overview

The document uses the term "entity" to refer to the target of the attestation token. The claims defined in this document are claims about an entity.

An entity is an implementation in hardware, software or both.

An entity is the same as the Attester Target Environment defined in RATS Architecture.

An entity also corresponds to a "system component" as defined in the Internet Security Glossary [RFC4949]. That glossary also defines "entity" and "system entity" as something that may be a person or organization as well as a system component. Here "entity" never refers to a person or organization.

An entity is never a server or a service.

An entity may be the whole device or it may be a subsystem, a subsystem of a subsystem and so on. EAT allows claims to be organized into submodules, nested EATs and so on. See Section 3.25. The entity to which a claim applies is the submodule in which it appears, or to the top-level entity if it doesn't appear in a submodule.

Some examples of entities:

- o A Secure Element
- o A TEE
- o A card in a network router
- o A network router, perhaps with each card in the router a submodule
- o An IoT device
- o An individual process
- o An app on a smartphone
- o A smartphone with many submodules for its many subsystems
- o A subsystem in a smartphone like the modem or the camera

An entity may have strong security like defenses against hardware invasive attacks. It may also have low security, having no special security defenses. There is no minimum security requirement to be an entity.

1.2. CWT, JWT, UCCS, UJCS and DEB

An EAT is a claims set about an entity based on one of the following:

- o CBOR Web Token (CWT) [RFC8392]
- o Unprotected CWT Claims Sets (UCCS) [UCCS.Draft]
- o JSON Web Token (JWT) [RFC7519]

All definitions, requirements, creation and validation procedures, security considerations, IANA registrations and so on from these carry over to EAT.

This specification extends those specifications by defining additional claims for attestation. This specification also describes the notion of a "profile" that can narrow the definition of an EAT, ensure interoperability and fill in details for specific usage scenarios. This specification also adds some considerations for registration of future EAT-related claims.

The identification of a protocol element as an EAT, whether CBOR or JSON encoded, follows the general conventions used by CWT, JWT and

UCCS. Largely this depends on the protocol carrying the EAT. In some cases it may be by content type (e.g., MIME type). In other cases it may be through use of CBOR tags. There is no fixed mechanism across all use cases.

This specification adds two more top-level messages:

- o Unprotected JWT Claims Set (UJCS) Section 4
- o Detached EAT Bundle (DEB), Section 5

A DEB is structure to hold a collection of detached claims sets and the EAT that separately provides integrity and authenticity protection for them. It can be either CBOR or JSON encoded.

1.3. CDDL, CBOR and JSON

This document defines Concise Binary Object Representation (CBOR) [RFC8949] and Javascript Object Notation (JSON) [RFC7159] encoding for an EAT. All claims in an EAT MUST use the same encoding except where explicitly allowed. It is explicitly allowed for a nested token to be of a different encoding. Some claims explicitly contain objects and messages that may use a different encoding than the enclosing EAT.

This specification uses Concise Data Definition Language (CDDL) [RFC8610] for all definitions. The implementor interprets the CDDL to come to either the CBOR or JSON encoding. In the case of JSON, Appendix E of [RFC8610] is followed. Additional rules are given in Section 8.2.2 where Appendix E is insufficient.

The CWT and JWT specifications were authored before CDDL was available and did not use CDDL. This specification includes a CDDL definition of most of what is defined in [RFC8392]. Similarly, this specification includes CDDL for most of what is defined in [RFC7519].

The UCCS specification does not include CDDL. This specification provides CDDL for it.

1.4. Operating Model and RATS Architecture

While it is not required that EAT be used with the RATS operational model described in Figure 1 in [RATS.Architecture], or even that it be used for attestation, this document is oriented around that model.

To summarize, an Attester generates Attestation Evidence. Attestation Evidence is a claims set describing various characteristics of an entity. Attestation Evidence also is usually

signed by a key that proves the entity and the evidence it produces are authentic. The claims set includes a nonce or some other means to provide freshness. EAT is designed to carry Attestation Evidence. The Attestation Evidence goes to a Verifier where the signature is verified. Some of the claims may also be checked against Reference Values. The Verifier then produces Attestation Results which is also usually a claims set. EAT is also designed to carry Attestation Results. The Attestation Results go to the Relying Party which is the ultimate consumer of the Remote Attestation Procedure. The Relying Party uses the Attestation Results as needed for the use case, perhaps allowing an entity on the network, allowing a financial transaction or such.

Note that sometimes the Verifier and Relying Party are not separate and thus there is no need for a protocol to carry Attestation Results.

1.4.1. Relationship between Attestation Evidence and Attestation Results

Any claim defined in this document or in the IANA CWT or JWT registry may be used in Attestation Evidence or Attestation Results.

Many claims in Attestation Evidence simply will pass through the Verifier to the Relying Party without modification. They will be verified as authentic from the entity by the Verifier just through normal verification of the Attester's signature. The UEID, Section 3.4, and Location, Section 3.15, are examples of claims that may be passed through.

Some claims in Attestation Evidence will be verified by the Verifier by comparison to Reference Values. These claims will not likely be conveyed to the Relying Party. Instead, some claim indicating they were checked may be added to the Attestation Results or it may be tacitly known that the Verifier always does this check. For example, the Verifier receives the Software Evidence claim, Section 3.23, compares it to Reference Values and conveys the results to the Relying Party in a Software Measurement Results Claim, Section 3.24.

In some cases the Verifier may provide privacy-preserving functionality by stripping or modifying claims that do not possess sufficient privacy-preserving characteristics. For example, the data in the Location claim, Section 3.15, may be modified to have a precision of a few kilometers rather than a few meters.

2. 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 BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

This document reuses terminology from JWT [RFC7519] and CWT [RFC8392].

Claim: A piece of information asserted about a subject. A claim is represented as pair with a value and either a name or key to identify it.

Claim Name: A unique text string that identifies the claim. It is used as the claim name for JSON encoding.

Claim Key: The CBOR map key used to identify a claim.

Claim Value: The value portion of the claim. A claim value can be any CBOR data item or JSON value.

CWT/JWT Claims Set: The CBOR map or JSON object that contains the claims conveyed by the CWT or JWT.

This document reuses terminology from RATS Architecture [RATS.Architecture]

Attester: A role performed by an entity (typically a device) whose Evidence must be appraised in order to infer the extent to which the Attester is considered trustworthy, such as when deciding whether it is authorized to perform some operation.

Verifier: A role that appraises the validity of Attestation Evidence about an Attester and produces Attestation Results to be used by a Relying Party.

Relying Party: A role that depends on the validity of information about an Attester, for purposes of reliably applying application specific actions. Compare /relying party/ in [RFC4949].

Attestation Evidence: A Claims Set generated by an Attester to be appraised by a Verifier. Attestation Evidence may include configuration data, measurements, telemetry, or inferences.

Attestation Results: The output generated by a Verifier, typically including information about an Attester, where the Verifier vouches for the validity of the results

Reference Values: A set of values against which values of Claims can be compared as part of applying an Appraisal Policy for Attestation Evidence. Reference Values are sometimes referred to in other documents as known-good values, golden measurements, or nominal values, although those terms typically assume comparison for equality, whereas here Reference Values might be more general and be used in any sort of comparison.

3. The Claims

This section describes new claims defined for attestation that are to be added to the CWT [IANA.CWT.Claims] and JWT [IANA.JWT.Claims] IANA registries.

This section also describes how several extant CWT and JWT claims apply in EAT.

CDDL, along with a text description, is used to define each claim independent of encoding. Each claim is defined as a CDDL group. In Section 8 on encoding, the CDDL groups turn into CBOR map entries and JSON name/value pairs.

Each claim described has a unique text string and integer that identifies it. CBOR encoded tokens MUST use only the integer for Claim Keys. JSON encoded tokens MUST use only the text string for Claim Names.

3.1. Token ID Claim (cti and jti)

CWT defines the "cti" claim. JWT defines the "jti" claim. These are equivalent to each other in EAT and carry a unique token identifier as they do in JWT and CWT. They may be used to defend against re use of the token but are distinct from the nonce that is used by the Relying Party to guarantee freshness and defend against replay.

3.2. Timestamp claim (iat)

The "iat" claim defined in CWT and JWT is used to indicate the date-of-creation of the token, the time at which the claims are collected and the token is composed and signed.

The data for some claims may be held or cached for some period of time before the token is created. This period may be long, even days. Examples are measurements taken at boot or a geographic

position fix taken the last time a satellite signal was received. There are individual timestamps associated with these claims to indicate their age is older than the "iat" timestamp.

CWT allows the use floating-point for this claim. EAT disallows the use of floating-point. An EAT token MUST NOT contain an iat claim in float-point format. Any recipient of a token with a floating-point format iat claim MUST consider it an error. A 64-bit integer representation of epoch time can represent a range of +/- 500 billion years, so the only point of a floating-point timestamp is to have precession greater than one second. This is not needed for EAT.

3.3. Nonce Claim (nonce)

All EATs should have a nonce to prevent replay attacks. The nonce is generated by the Relying Party, the end consumer of the token. It is conveyed to the entity over whatever transport is in use before the token is generated and then included in the token as the nonce claim.

This documents the nonce claim for registration in the IANA CWT claims registry. This is equivalent to the JWT nonce claim that is already registered.

The nonce must be at least 8 bytes (64 bits) long as fewer bytes are unlikely to be secure. A maximum of 64 bytes is set to limit the memory a constrained implementation uses. This size range is not set for the already-registered JWT nonce, but it should follow this size recommendation when used in an EAT.

Multiple nonces are allowed to accommodate multistage verification and consumption.

```
$$claims-set-claims //=  
    (nonce-label => nonce-type / [ 2* nonce-type ])  
  
nonce-type = bstr .size (8..64)
```

3.4. Universal Entity ID Claim (ueid)

A UEID identifies an individual manufactured entity like a mobile phone, a water meter, a Bluetooth speaker or a networked security camera. It may identify the entire entity or a submodule. It does not identify types, models or classes of entities. It is akin to a serial number, though it does not have to be sequential.

UEIDs MUST be universally and globally unique across manufacturers and countries. UEIDs MUST also be unique across protocols and systems, as tokens are intended to be embedded in many different

protocols and systems. No two products anywhere, even in completely different industries made by two different manufacturers in two different countries should have the same UEID (if they are not global and universal in this way, then Relying Parties receiving them will have to track other characteristics of the entity to keep entities distinct between manufacturers).

There are privacy considerations for UEIDs. See Section 10.1.

The UEID is permanent. It MUST never change for a given entity.

A UEID is constructed of a single type byte followed by the bytes that are the identifier. Several types are allowed to accommodate different industries, different manufacturing processes and to have an alternative that doesn't require paying a registration fee.

Creation of new types requires a Standards Action [RFC8126].

UEIDs are variable length. All implementations MUST be able to receive UEIDs that are 33 bytes long (1 type byte and 256 bits). No UEID longer than 33 bytes SHOULD be sent.

Type Byte	Type Name	Specification
0x01	RAND	This is a 128, 192 or 256-bit random number generated once and stored in the entity. This may be constructed by concatenating enough identifiers to make up an equivalent number of random bits and then feeding the concatenation through a cryptographic hash function. It may also be a cryptographic quality random number generated once at the beginning of the life of the entity and stored. It MUST NOT be smaller than 128 bits. See the length analysis in Appendix B.
0x02	IEEE EUI	This uses the IEEE company identification registry. An EUI is either an EUI-48, EUI-60 or EUI-64 and made up of an OUI, OUI-36 or a CID, different registered company identifiers, and some unique per-entity identifier. EUIs are often the same as or similar to MAC addresses. This type includes MAC-48, an obsolete name for EUI-48. (Note that while entities with multiple network interfaces may have multiple MAC addresses, there is only one UEID for an entity) [IEEE.802-2001], [OUI.Guide].
0x03	IMEI	This is a 14-digit identifier consisting of an 8-digit Type Allocation Code and a 6-digit serial number allocated by the manufacturer, which SHALL be encoded as byte string of length 14 with each byte as the digit's value (not the ASCII encoding of the digit; the digit 3 encodes as 0x03, not 0x33). The IMEI value encoded SHALL NOT include Luhn checksum or SVN information. See [ThreeGPP.IMEI].

Table 1: UEID Composition Types

UEIDs are not designed for direct use by humans (e.g., printing on the case of a device), so no textual representation is defined.

The consumer (the Relying Party) of a UEID MUST treat a UEID as a completely opaque string of bytes and not make any use of its internal structure. For example, they should not use the OUI part of a type 0x02 UEID to identify the manufacturer of the entity. Instead, they should use the OEMID claim. See Section 3.6. The reasons for this are:

- o UEIDs types may vary freely from one manufacturer to the next.

- o New types of UEIDs may be created. For example, a type 0x07 UEID may be created based on some other manufacturer registration scheme.
- o Entity manufacturers are allowed to change from one type of UEID to another anytime they want. For example, they may find they can optimize their manufacturing by switching from type 0x01 to type 0x02 or vice versa. The essential requirement on the manufacturer is that UEIDs be universally unique.

A Device Identifier URN is registered for UEIDs. See Section 9.3.4.

```
$$claims-set-claims // = (ueid-label => ueid-type)
```

```
ueid-type = bstr .size (7..33)
```

3.5. Semi-permanent UEIDs (SUEIDs)

An SEUID is of the same format as a UEID, but it MAY change to a different value on device life-cycle events. Examples of these events are change of ownership, factory reset and on-boarding into an IoT device management system. An entity MAY have both a UEID and SUEIDs, neither, one or the other.

There MAY be multiple SUEIDs. Each one has a text string label the purpose of which is to distinguish it from others in the token. The label MAY name the purpose, application or type of the SUEID. Typically, there will be few SUEIDs so there is no need for a formal labeling mechanism like a registry. The EAT profile MAY describe how SUEIDs should be labeled. If there is only one SUEID, the claim remains a map and there still must be a label. For example, the label for the SUEID used by FIDO Onboarding Protocol could simply be "FIDO".

There are privacy considerations for SUEIDs. See Section 10.1.

A Device Identifier URN is registered for SUEIDs. See Section 9.3.4.

```
$$claims-set-claims // = (sueids-label => sueids-type)
```

```
sueids-type = {
  + tstr => ueid-type
}
```

3.6. Hardware OEM Identification (oemid)

This claim identifies the Original Equipment Manufacturer (OEM) of the hardware. Any of the three forms described below MAY be used at the convenience of the claim sender. The receiver of this claim MUST be able to handle all three forms.

3.6.1. Random Number Based OEMID

The random number based OEMID MUST always 16 bytes (128 bits).

The OEM MAY create their own ID by using a cryptographic-quality random number generator. They would perform this only once in the life of the company to generate the single ID for said company. They would use that same ID in every entity they make. This uniquely identifies the OEM on a statistical basis and is large enough should there be ten billion companies.

The OEM MAY also use a hash function like SHA-256 and truncate the output to 128 bits. The input to the hash should be somethings that have at least 96 bits of entropy, but preferably 128 bits of entropy. The input to the hash MAY be something whose uniqueness is managed by a central registry like a domain name.

In JSON format tokens this MUST be base64url encoded.

3.6.2. IEEE Based OEMID

The IEEE operates a global registry for MAC addresses and company IDs. This claim uses that database to identify OEMs. The contents of the claim may be either an IEEE MA-L, MA-M, MA-S or an IEEE CID [IEEE.RA]. An MA-L, formerly known as an OUI, is a 24-bit value used as the first half of a MAC address. MA-M similarly is a 28-bit value uses as the first part of a MAC address, and MA-S, formerly known as OUI-36, a 36-bit value. Many companies already have purchased one of these. A CID is also a 24-bit value from the same space as an MA-L, but not for use as a MAC address. IEEE has published Guidelines for Use of EUI, OUI, and CID [OUI.Guide] and provides a lookup service [OUI.Lookup].

Companies that have more than one of these IDs or MAC address blocks SHOULD select one and prefer that for all their entities.

Commonly, these are expressed in Hexadecimal Representation as described in [IEEE.802-2001]. It is also called the Canonical format. When this claim is encoded the order of bytes in the bstr are the same as the order in the Hexadecimal Representation. For

example, an MA-L like "AC-DE-48" would be encoded in 3 bytes with values 0xAC, 0xDE, 0x48.

This format is always 3 bytes in size in CBOR.

In JSON format tokens, this MUST be base64url encoded and always 4 bytes.

3.6.3. IANA Private Enterprise Number Based OEMID

IANA maintains a integer-based company registry called the Private Enterprise Number (PEN) [PEN].

PENs are often used to create an OID. That is not the case here. They are used only as an integer.

In CBOR this value MUST be encoded as a major type 0 integer and is typically 3 bytes. In JSON, this value MUST be encoded as a number.

```
oemid-pen = int
```

```
oemid-ieee = bstr .size 3
```

```
oemid-random = bstr .size 16
```

```
$$claims-set-claims //= (  
  oemid-label =>  
    oemid-random / oemid-ieee / oemid-pen  
)
```

3.7. Hardware Model Claim (hardware-model)

This claim differentiates hardware models, products and variants manufactured by a particular OEM, the one identified by OEM ID in Section 3.6.

This claim must be unique so as to differentiate the models and products for the OEM ID. This claim does not have to be globally unique, but it can be. A receiver of this claim MUST not assume it is globally unique. To globally identify a particular product, the receiver should concatenate the OEM ID and this claim.

The granularity of the model identification is for each OEM to decide. It may be very granular, perhaps including some version information. It may be very general, perhaps only indicating top-level products.

The purpose of this claim is to identify models within protocols, not for human-readable descriptions. The format and encoding of this claim should not be human-readable to discourage use other than in protocols. If this claim is to be derived from an already-in-use human-readable identifier, it can be run through a hash function.

There is no minimum length so that an OEM with a very small number of models can use a one-byte encoding. The maximum length is 32 bytes. All receivers of this claim MUST be able to receive this maximum size.

The receiver of this claim MUST treat it as a completely opaque string of bytes, even if there is some apparent naming or structure. The OEM is free to alter the internal structure of these bytes as long as the claim continues to uniquely identify its models.

```
hardware-model-type = bytes .size (1..32)
```

```
$$claims-set-claims // = (
    hardware-model-label => hardware-model-type
)
```

3.8. Hardware Version Claims (hardware-version-claims)

The hardware version is a text string the format of which is set by each manufacturer. The structure and sorting order of this text string can be specified using the version-scheme item from CoSWID [CoSWID]. It is useful to know how to sort versions so the newer can be distinguished from the older.

The hardware version can also be given by a 13-digit [EAN-13]. A new CoSWID version scheme is registered with IANA by this document in Section 9.3.3. An EAN-13 is also known as an International Article Number or most commonly as a bar code.

```
$$claims-set-claims // = (
    hardware-version-label => hardware-version-type
)
```

```
hardware-version-type = [
    version:  tstr,
    scheme:   $version-scheme
]
```

3.9. Software Name Claim

This is a free-form text claim for the name of the software for the entity or submodule. A CoSWID manifest or other type of manifest can be used instead if this claim is too limited to correctly characterize the SW for the entity or submodule.

```
$$claims-set-claims //= ( sw-name-label => tstr )
```

3.10. Software Version Claim

This makes use of the CoSWID version scheme data type to give a simple version for the software. A full CoSWID manifest or other type of manifest can be used instead if this is too simple.

```
$$claims-set-claims //= (sw-version-label => sw-version-type)
```

```
sw-version-type = [  
    version:  tstr,  
    scheme:   $version-scheme ; As defined by CoSWID  
]
```

3.11. The Security Level Claim (security-level)

This claim characterizes the entity's ability to defend against attacks aimed at capturing the signing key, forging claims and at forging EATs. This is by defining four security levels.

This claim describes the security environment and countermeasures available on the entity where the attestation key resides and the claims originate.

- 1 - Unrestricted: There is some expectation that implementor will protect the attestation signing keys at this level. Otherwise, the EAT provides no meaningful security assurances.
- 2 - Restricted: Entities at this level are not general-purpose operating environments that host features, such as app download systems, web browsers and complex applications. It is akin to the secure-restricted level (see below) without the security orientation. Examples include a Wi-Fi subsystem, an IoT camera, or sensor device. Often these can be considered more secure than unrestricted just because they are much simpler and a smaller attack surface, but this won't always be the case. Some unrestricted devices may be implemented in a way that provides poor protection of signing keys.

- 3 - Secure-Restricted: Entities at this level must meet the criteria defined in Section 4 of FIDO Allowed Restricted Operating Environments [FIDO.AROE]. Examples include TEE's and schemes using virtualization-based security. Security at this level is aimed at defending against large-scale network/remote attacks against the entity.
- 4 - Hardware: Entities at this level must include substantial defense against physical or electrical attacks against the entity itself. It is assumed the potential attacker has captured the entity and can disassemble it. Examples include TPMs and Secure Elements.

The entity should claim the highest security level it achieves and no higher. This set is not extensible so as to provide a common interoperable description of security level to the Relying Party. If a particular use case considers this claim to be inadequate, it can define its own proprietary claim. It may consider including both this claim as a coarse indication of security and its own proprietary claim as a refined indication.

This claim is not intended as a replacement for a formal security certification scheme, such as those based on FIPS 140 [FIPS-140] or those based on Common Criteria [Common.Criteria]. See Section 3.21.

```
$$claims-set-claims // = (
    security-level-label =>
        security-level-chor-type /
        security-level-json-type
)
```

```
security-level-chor-type = &(
    unrestricted: 1,
    restricted: 2,
    secure-restricted: 3,
    hardware: 4
)
```

```
security-level-json-type =
    "unrestricted" /
    "restricted" /
    "secure-restricted" /
    "hardware"
```


3.12. Secure Boot Claim (secure-boot)

The value of true indicates secure boot is enabled. Secure boot is considered enabled when the firmware and operating system, are under control of the manufacturer of the entity identified in the OEMID claim described in Section 3.6. Control by the manufacturer of the firmware and the operating system may be by it being in ROM, being cryptographically authenticated, a combination of the two or similar.

```
$$claims-set-claims //=(secure-boot-label => bool)
```

3.13. Debug Status Claim (debug-status)

This applies to entity-wide or submodule-wide debug facilities of the entity like JTAG and diagnostic hardware built into chips. It applies to any software debug facilities related to root, operating system or privileged software that allow system-wide memory inspection, tracing or modification of non-system software like user mode applications.

This characterization assumes that debug facilities can be enabled and disabled in a dynamic way or be disabled in some permanent way such that no enabling is possible. An example of dynamic enabling is one where some authentication is required to enable debugging. An example of permanent disabling is blowing a hardware fuse in a chip. The specific type of the mechanism is not taken into account. For example, it does not matter if authentication is by a global password or by per-entity public keys.

As with all claims, the absence of the debug level claim means it is not reported. A conservative interpretation might assume the enabled state.

This claim is not extensible so as to provide a common interoperable description of debug status. If a particular implementation considers this claim to be inadequate, it can define its own proprietary claim. It may consider including both this claim as a coarse indication of debug status and its own proprietary claim as a refined indication.

The higher levels of debug disabling requires that all debug disabling of the levels below it be in effect. Since the lowest level requires that all of the target's debug be currently disabled, all other levels require that too.

There is no inheritance of claims from a submodule to a superior module or vice versa. There is no assumption, requirement or guarantee that the target of a superior module encompasses the

targets of submodules. Thus, every submodule must explicitly describe its own debug state. The receiver of an EAT MUST not assume that debug is turned off in a submodule because there is a claim indicating it is turned off in a superior module.

An entity may have multiple debug facilities. The use of plural in the description of the states refers to that, not to any aggregation or inheritance.

The architecture of some chips or devices may be such that a debug facility operates for the whole chip or device. If the EAT for such a chip includes submodules, then each submodule should independently report the status of the whole-chip or whole-device debug facility. This is the only way the receiver can know the debug status of the submodules since there is no inheritance.

3.13.1. Enabled

If any debug facility, even manufacturer hardware diagnostics, is currently enabled, then this level must be indicated.

3.13.2. Disabled

This level indicates all debug facilities are currently disabled. It may be possible to enable them in the future. It may also be that they were enabled in the past, but they are currently disabled.

3.13.3. Disabled Since Boot

This level indicates all debug facilities are currently disabled and have been so since the entity booted/started.

3.13.4. Disabled Permanently

This level indicates all non-manufacturer facilities are permanently disabled such that no end user or developer can enable them. Only the manufacturer indicated in the OEMID claim can enable them. This also indicates that all debug facilities are currently disabled and have been so since boot/start.

3.13.5. Disabled Fully and Permanently

This level indicates that all debug facilities for the entity are permanently disabled.

```
$$claims-set-claims // = (
    debug-status-label =>
        debug-status-cbor-type / debug-status-json-type
)

debug-status-cbor-type = &(
    enabled: 0,
    disabled: 1,
    disabled-since-boot: 2,
    disabled-permanently: 3,
    disabled-fully-and-permanently: 4
)

debug-status-json-type =
    "enabled" /
    "disabled" /
    "disabled-since-boot" /
    "disabled-permanently" /
    "disabled-fully-and-permanently"
```

3.14. Including Keys

An EAT may include a cryptographic key such as a public key. The signing of the EAT binds the key to all the other claims in the token.

The purpose for inclusion of the key may vary by use case. For example, the key may be included as part of an IoT device onboarding protocol. When the FIDO protocol includes a public key in its attestation message, the key represents the binding of a user, device and Relying Party. This document describes how claims containing keys should be defined for the various use cases. It does not define specific claims for specific use cases.

Keys in CBOR format tokens SHOULD be the COSE_Key format [RFC8152] and keys in JSON format tokens SHOULD be the JSON Web Key format [RFC7517]. These two formats support many common key types. Their use avoids the need to decode other serialization formats. These two formats can be extended to support further key types through their IANA registries.

The general confirmation claim format [RFC8747], [RFC7800] may also be used. It provides key encryption. It also allows for inclusion by reference through a key ID. The confirmation claim format may be employed in the definition of some new claim for a particular use case.

When the actual confirmation claim is included in an EAT, this document associates no use case semantics other than proof of possession. Different EAT use cases may choose to associate further semantics. The key in the confirmation claim **MUST** be protected in the same way as the key used to sign the EAT. That is, the same, equivalent or better hardware defenses, access controls, key generation and such must be used.

3.15. The Location Claim (location)

The location claim gives the location of the entity from which the attestation originates. It is derived from the W3C Geolocation API [W3C.GeoLoc]. The latitude, longitude, altitude and accuracy must conform to [WGS84]. The altitude is in meters above the [WGS84] ellipsoid. The two accuracy values are positive numbers in meters. The heading is in degrees relative to true north. If the entity is stationary, the heading is NaN (floating-point not-a-number). The speed is the horizontal component of the entity velocity in meters per second.

When encoding floating-point numbers half-precision **SHOULD NOT** be used. They usually do not provide enough precision for a geographic location.

The location may have been cached for a period of time before token creation. For example, it might have been minutes or hours or more since the last contact with a GPS satellite. Either the timestamp or age data item can be used to quantify the cached period. The timestamp data item is preferred as it a non-relative time.

The age data item can be used when the entity doesn't know what time it is either because it doesn't have a clock or it isn't set. The entity **MUST** still have a "ticker" that can measure a time interval. The age is the interval between acquisition of the location data and token creation.

See location-related privacy considerations in Section 10.2.

```
$$claims-set-claims // = (location-label => location-type)
```

```
location-type = {  
    latitude => number,  
    longitude => number,  
    ? altitude => number,  
    ? accuracy => number,  
    ? altitude-accuracy => number,  
    ? heading => number,  
    ? speed => number,  
    ? timestamp => ~time-int,  
    ? age => uint  
}  
  
latitude = 1 / "latitude"  
longitude = 2 / "longitude"  
altitude = 3 / "altitude"  
accuracy = 4 / "accuracy"  
altitude-accuracy = 5 / "altitude-accuracy"  
heading = 6 / "heading"  
speed = 7 / "speed"  
timestamp = 8 / "timestamp"  
age = 9 / "age"
```

3.16. The Uptime Claim (uptime)

The "uptime" claim MUST contain a value that represents the number of seconds that have elapsed since the entity or submod was last booted.

```
$$claims-set-claims // = (uptime-label => uint)
```

3.17. The Boot Odometer Claim (odometer)

The "odometer" claim contains a value that represents the number of times the entity or submod has been booted. Support for this claim requires a persistent storage on the device.

```
$$claims-set-claims // = (odometer-label => uint)
```

3.18. The Boot Seed Claim (boot-seed)

The Boot Seed claim MUST contain a random value created at system boot time that will allow differentiation of reports from different boot sessions.

This value is usually public. It is not a secret and MUST NOT be used for any purpose that a secret seed is needed, such as seeding a random number generator.

`$$claims-set-claims // = (boot-seed-label => bytes)`

3.19. The Intended Use Claim (intended-use)

EAT's may be used in the context of several different applications. The intended-use claim provides an indication to an EAT consumer about the intended usage of the token. This claim can be used as a way for an application using EAT to internally distinguish between different ways it uses EAT.

- 1 - Generic: Generic attestation describes an application where the EAT consumer requires the most up-to-date security assessment of the attesting entity. It is expected that this is the most commonly-used application of EAT.
- 2- Registration: Entities that are registering for a new service may be expected to provide an attestation as part of the registration process. This intended-use setting indicates that the attestation is not intended for any use but registration.
- 3 - Provisioning: Entities may be provisioned with different values or settings by an EAT consumer. Examples include key material or device management trees. The consumer may require an EAT to assess entity security state of the entity prior to provisioning.
- 4 - Certificate Issuance Certification Authorities (CA's) may require attestations prior to the issuance of certificates related to keypairs hosted at the entity. An EAT may be used as part of the certificate signing request (CSR).
- 5 - Proof-of-Possession: An EAT consumer may require an attestation as part of an accompanying proof-of-possession (PoP) application. More precisely, a PoP transaction is intended to provide to the recipient cryptographically-verifiable proof that the sender has possession of a key. This kind of attestation may be necessary to verify the security state of the entity storing the private key used in a PoP application.

```
$$claims-set-claims //= (
  intended-use-label =>
    intended-use-cbor-type / intended-use-json-type
)

intended-use-cbor-type = &(
  generic: 1,
  registration: 2,
  provisioning: 3,
  csr: 4,
  pop: 5
)

intended-use-json-type =
  "generic" /
  "registration" /
  "provisioning" /
  "csr" /
  "pop"
```

3.20. The Profile Claim (profile)

See Section 7 for the detailed description of a profile.

A profile is identified by either a URL or an OID. Typically, the URI will reference a document describing the profile. An OID is just a unique identifier for the profile. It may exist anywhere in the OID tree. There is no requirement that the named document be publicly accessible. The primary purpose of the profile claim is to uniquely identify the profile even if it is a private profile.

The OID is always absolute and never relative. In CBOR tokens, the OID MUST be encoded according to [RFC9090] and the URI according to [RFC8949]. Both are unwrapped and thus not CBOR tags. In JSON tokens, the OID is a string of the form "X.X.X", and a URI is a normal URI string.

Note that this is named "eat_profile" for JWT and is distinct from the already registered "profile" claim in the JWT claims registry.

```
$$claims-set-claims //= (profile-label => ~uri / ~oid)
```

3.21. The DLOA (Digital Letter or Approval) Claim (dloas)

A DLOA (Digital Letter of Approval) [DLOA] is an XML document that describes a certification that an entity has received. Examples of certifications represented by a DLOA include those issued by Global Platform and those based on Common Criteria. The DLOA is unspecific

to any particular certification type or those issued by any particular organization.

This claim is typically issued by a Verifier, not an Attester. When this claim is issued by a Verifier, it MUST be because the entity has received the certification in the DLOA.

This claim MAY contain more than one DLOA. If multiple DLOAs are present, it MUST be because the entity received all of the certifications.

DLOA XML documents are always fetched from a registrar that stores them. This claim contains several data items used to construct a URL for fetching the DLOA from the particular registrar.

This claim MUST be encoded as an array with either two or three elements. The first element MUST be the URI for the registrar. The second element MUST be a platform label indicating which platform was certified. If the DLOA applies to an application, then the third element is added which MUST be an application label. The method of constructing the registrar URI, platform label and possibly application label is specified in [DLOA].

```
$$claims-set-claims // = (  
    dloas-label => [ + dloa-type ]  
)
```

```
dloa-type = [  
    dloa_registrar: ~uri  
    dloa_platform_label: text  
    ? dloa_application_label: text  
]
```

3.22. The Software Manifests Claim (manifests)

This claim contains descriptions of software present on the entity. These manifests are installed on the entity when the software is installed or are created as part of the installation process. Installation is anything that adds software to the entity, possibly factory installation, the user installing elective applications and so on. The defining characteristic is they are created by the software manufacturer. The purpose of these claims in an EAT is to relay them without modification to the Verifier and possibly to the Relying Party.

Some manifests may be signed by their software manufacturer before they are put into this EAT claim. When such manifests are put into this claim, the manufacturer's signature SHOULD be included. For

example, the manifest might be a CoSWID signed by the software manufacturer, in which case the full signed CoSWID should be put in this claim.

This claim allows multiple formats for the manifest. For example, the manifest may be a CBOR-format CoSWID, an XML-format SWID or other. Identification of the type of manifest is always by a CBOR tag. In many cases, for examples CoSWID, a tag will already be registered with IANA. If not, a tag MUST be registered. It can be in the first-come-first-served space which has minimal requirements for registration.

The claim is an array of one or more manifests. To facilitate hand off of the manifest to a decoding library, each manifest is contained in a byte string. This occurs for CBOR-format manifests as well as non-CBOR format manifests.

If a particular manifest type uses CBOR encoding, then the item in the array for it MUST be a byte string that contains a CBOR tag. The EAT decoder must decode the byte string and then the CBOR within it to find the tag number to identify the type of manifest. The contents of the byte string is then handed to the particular manifest processor for that type of manifest. CoSWID and SUIT manifest are examples of this.

If a particular manifest type does not use CBOR encoding, then the item in the array for it MUST be a CBOR tag that contains a byte string. The EAT decoder uses the tag to identify the processor for that type of manifest. The contents of the tag, the byte string, are handed to the manifest processor. Note that a byte string is used to contain the manifest whether it is a text based format or not. An example of this is an XML format ISO/IEC 19770 SWID.

It is not possible to describe the above requirements in CDDL, so the type for an individual manifest is any in the CDDL below. The above text sets the encoding requirement.

This claim allows for multiple manifests in one token since multiple software packages are likely to be present. The multiple manifests MAY be of multiple formats. In some cases EAT submodules may be used instead of the array structure in this claim for multiple manifests.

When the [CoSWID] format is used, it MUST be a payload CoSWID, not an evidence CoSWID.

```
$$claims-set-claims //= (  
    manifests-label => manifests-type  
)  
  
manifests-type = [+ $$manifest-formats]  
  
coswid-that-is-a-cbor-tag-xx = tagged-coswid<concise-swid-tag>  
  
$$manifest-formats /= bytes .cbor coswid-that-is-a-cbor-tag-xx
```

3.23. The Software Evidence Claim (swevidence)

This claim contains descriptions, lists, evidence or measurements of the software that exists on the entity. The defining characteristic of this claim is that its contents are created by processes on the entity that inventory, measure or otherwise characterize the software on the entity. The contents of this claim do not originate from the software manufacturer.

This claim uses the same mechanism for identification of the type of the swevidence as is used for the type of the manifest in the manifests claim. It also uses the same byte string based mechanism for containing the claim and easing the hand off to a processing library. See the discussion above in the manifests claim.

When the [CoSWID] format is used, it MUST be evidence CoSWIDs, not payload CoSWIDS.

```
$$claims-set-claims //= (  
    swevidence-label => swevidence-type  
)  
  
swevidence-type = [+ $$swevidence-formats]  
  
coswid-that-is-a-cbor-tag = tagged-coswid<concise-swid-tag>  
$$swevidence-formats /= bytes .cbor coswid-that-is-a-cbor-tag
```

3.24. The SW Measurement Results Claim (swresults)

This claims reports the outcome of the comparison of a measurement on some software to the expected Reference Values. It may report a successful comparison, failed comparison or other.

This claim MAY be generated by the Verifier and sent to the Relying Party. For example, it could be the results of the Verifier comparing the contents of the swevidence claim to Reference Values.

This claim MAY also be generated on the entity if the entity has the ability for one subsystem to measure another subsystem. For example, a TEE might have the ability to measure the software of the rich OS and may have the Reference Values for the rich OS.

Within an attestation target or submodule, multiple results can be reported. For example, it may be desirable to report the results for the kernel and each individual application separately.

For each software objective, the following can be reported. TODO:
defined objective

3.24.1. Scheme

This is the free-form text name of the verification system or scheme that performed the verification. There is no official registry of schemes or systems. It may be the name of a commercial product or such.

3.24.2. Objective

This roughly characterizes the coverage of the software measurement software. This corresponds to the attestation target or the submodule. If all of the indicated target is not covered, the measurement must indicate partial.

- 1 - all: Indicates all the software has been verified, for example, all the software in the attestation target or the submodule
- 2 - firmware: Indicates all of and only the firmware
- 3 - kernel: Refers to all of the most-privileged software, for example the Linux kernel
- 4 - privileged: Refers to all of the software used by the root, system or administrative account
- 5 - system-libs: Refers to all of the system libraries that are broadly shared and used by applications and such
- 6 - partial: Some other partial set of the software

3.24.3. Results

This describes the result of the measurement and also the comparison to Reference Values.

- 1 - verification-not-run: Indicates that no attempt was made to run the verification
- 2 - verification-indeterminate: The verification was attempted, but it did not produce a result; perhaps it ran out of memory, the battery died or such
- 3 - verification-failed: The verification ran to completion, the comparison was completed and did not compare correctly to the Reference Values
- 4 - fully-verified: The verification ran to completion and all measurements compared correctly to Reference Values
- 5 - partially-verified: The verification ran to completion and some, but not all, measurements compared correctly to Reference Values

3.24.4. Objective Name

This is a free-form text string that describes the objective. For example, "Linux kernel" or "Facebook App"

```
$$claims-set-claims // = (swresults-label => [ + swresult-type ])  
  
verification-result-cbor-type = &(  
    verification-not-run: 1,  
    verification-indeterminate: 2,  
    verification-failed: 3,  
    fully-verified: 4,  
    partially-verified: 5,  
)  
  
verification-result-json-type =  
    "verification-not-run" /  
    "verification-indeterminate" /  
    "verification-failed" /  
    "fully-verified" /  
    "partially-verified"  
  
verification-objective-cbor-type = &(  
    all: 1,  
    firmware: 2,  
    kernel: 3,  
    privileged: 4,  
    system-libs: 5,  
    partial: 6,  
)  
  
verification-objective-json-type =  
    "all" /  
    "firmware" /  
    "kernel" /  
    "privileged" /  
    "system-libs" /  
    "partial"  
  
swresult-type = [  
    verification-system: tstr,  
    objective: verification-objective-cbor-type /  
        verification-objective-json-type,  
    result: verification-result-cbor-type /  
        verification-result-json-type,  
    ? objective-name: tstr  
]
```

3.25. Submodules (submods)

Some devices are complex, having many subsystems. A mobile phone is a good example. It may have several connectivity subsystems for communications (e.g., Wi-Fi and cellular). It may have subsystems for low-power audio and video playback. It may have multiple security-oriented subsystems like a TEE and a Secure Element.

The claims for a subsystem can be grouped together in a submodule or submod.

The submods are in a single map/object, one entry per submodule. There is only one submods map/object in a token. It is identified by its specific label. It is a peer to other claims, but it is not called a claim because it is a container for a claims set rather than an individual claim. This submods part of a token allows what might be called recursion. It allows claims sets inside of claims sets inside of claims sets...

3.25.1. Submodule Types

The following sections define the three types of submodules:

- o A submodule Claims-Set
- o A nested token, which can be any valid EAT token, CBOR or JSON
- o The digest of a detached Claims-Set

3.25.1.1. Submodule Claims-Set

This is a subordinate Claims-Set containing claims about the submodule.

The submodule claims-set is produced by the same Attester as the surrounding token. It is secured using the same mechanism as the enclosing token (e.g., it is signed by the same attestation key). It roughly corresponds to an Attester Target Environment, as described in the RATS architecture.

It may contain claims that are the same as its surrounding token or superior submodules. For example, the top-level of the token may have a UEID, a submod may have a different UEID and a further subordinate submodule may also have a UEID.

The encoding of a submodule Claims-Set MUST be the same as the encoding as the token it is part of.

This data type for this type of submodule is a map/object. It is identified when decoding by it's type being a map/object.

3.25.1.2. Nested Token

This type of submodule is a fully formed complete token. It is typically produced by a separate Attester. It is typically used by a Composite Device as described in RATS Architecture [RATS.Architecture] In being a submodule of the surrounding token, it is cryptographically bound to the surrounding token. If it was conveyed in parallel with the surrounding token, there would be no such binding and attackers could substitute a good attestation from another device for the attestation of an errant subsystem.

A nested token does not need to use the same encoding as the enclosing token. This is to allow Composite Devices to be built without regards to the encoding supported by their Attesters. Thus a CBOR-encoded token like a CWT or UCCS can have a JWT as a nested token submodule and a JSON-encoded token can have a CWT or UCCS as a nested token submodule.

The following two sections describe how to encode and decode a nested token.

3.25.1.2.1. Surrounding EAT is CBOR-Encoded

This describes the encoding and decoding of CBOR or JSON-encoded tokens nested inside a CBOR-encoded token.

If the nested token is CBOR-encoded, then it MUST be a CBOR tag and MUST be wrapped in a byte string. The tag identifies whether the nested token is a CWT, a UCCS, a CBOR-encoded DEB, or some other CBOR-format token defined in the future. A nested CBOR-encoded token that is not a CBOR tag is NOT allowed.

If the nested token is JSON-encoded, then the data item MUST be a text string. The text string MUST contain a JSON-encoded array of two items. The first item is a string identifying the type of the token. The second item is the JSON-encoded token.

The string identifying the JSON-encoded token MUST be one of the following:

"JWT": The second item MUST be a JWT formatted according to [RFC7519]

"UJCS": The second item MUST be a UJCS-Message as defined in this document.

"DEB": The second item MUST be a JSON-encoded Detached EAT Bundle as defined in this document.

The definition of additional types requires a standards action.

When decoding, if a byte string is encountered, it is known to be a nested CBOR-encoded token. The byte string wrapping is removed. The type of the token is determined by the CBOR tag.

When decoding, if a text string is encountered, it is known to be a JSON-encoded token. The two-item array is decoded and tells the type of the JSON-encoded token.

```
Nested-Token =  
    tstr / ; A JSON-encoded Nested-Token (see json-nested-token.cddl)  
    bstr .cbor Tagged-CBOR-Token
```

3.25.1.2.2. Surrounding EAT is JSON-Encoded

This describes the encoding and decoding of CBOR or JSON-encoded tokens nested inside a JSON-encoded token.

The nested token MUST be an array of two in the same format as described in the section above.

A CBOR-encoded token nested inside a JSON-encoded MUST use the same array of two, but with the type as follows:

"CBOR": Some base64url-encoded CBOR that is a tag, typically a CWT, UCCS or CBOR-encoded DEB

When decoding, the array of two is decoded. The first item indicates the type and encoding of the nested token. If the type string is not "CBOR", then the token is JSON-encoded and of the type indicated by the string.

If the type string is "CBOR", then the token is CBOR-encoded. The base64url encoding is removed. The CBOR-encoded data is then decoded. The type of nested token is determined by the CBOR-tag. It is an error if the CBOR is not a tag.


```
Nested-Token = [  
  type : "JWT" / "CBOR" / "UJCS" / "DEB",  
  nested-token : JWT-Message /  
                  B64URL-Tagged-CBOR-Token /  
                  DEB-JSON-Message /  
                  UJCS-Message  
]
```

```
B64URL-Tagged-CBOR-Token = tstr .regexp "[A-Za-z0-9_=-]+"
```

3.25.1.3. Detached Submodule Digest

This is type of submodule equivalent to a Claims-Set submodule, except the Claims-Set is conveyed separately outside of the token.

This type of submodule consists of a digest made using a cryptographic hash of a Claims-Set. The Claims-Set is not included in the token. It is conveyed to the Verifier outside of the token. The submodule containing the digest is called a detached digest. The separately conveyed Claims-Set is called a detached claims set.

The input to the digest is exactly the byte-string wrapped encoded form of the Claims-Set for the submodule. That Claims-Set can include other submodules including nested tokens and detached digests.

The primary use for this is to facilitate the implementation of a small and secure attester, perhaps purely in hardware. This small, secure attester implements COSE signing and only a few claims, perhaps just UEID and hardware identification. It has inputs for digests of submodules, perhaps 32-byte hardware registers. Software running on the device constructs larger claim sets, perhaps very large, encodes them and digests them. The digests are written into the small secure attesters registers. The EAT produced by the small secure attester only contains the UEID, hardware identification and digests and is thus simple enough to be implemented in hardware. Probably, every data item in it is of fixed length.

The integrity protection for the larger Claims Sets will not be as secure as those originating in hardware block, but the key material and hardware-based claims will be. It is possible for the hardware to enforce hardware access control (memory protection) on the digest registers so that some of the larger claims can be more secure. For example, one register may be writable only by the TEE, so the detached claims from the TEE will have TEE-level security.

The data type for this type of submodule MUST be an array. It contains two data items, an algorithm identifier and a byte string containing the digest.

When decoding a CBOR format token the detached digest type is distinguished from the other types by it being an array. In CBOR the none of other submodule types are arrays.

When decoding a JSON format token, a little more work is required because both the nested token and detached digest types are an array. To distinguish the nested token from the detached digest, the first element in the array is examined. If it is "JWT", "UJCS" or "DEB", the the submodule is a nested token. Otherwise it will contain an algorithm identifier and is a detached digest.

A DEB, described in Section 5, may be used to convey detached claims sets and the token with their detached digests. EAT, however, doesn't require use of a DEB. Any other protocols may be used to convey detached claims sets and the token with their detached digests. Note that since detached Claims-Sets are usually signed, protocols conveying them must make sure they are not modified in transit.

3.25.2. No Inheritance

The subordinate modules do not inherit anything from the containing token. The subordinate modules must explicitly include all of their claims. This is the case even for claims like the nonce.

This rule is in place for simplicity. It avoids complex inheritance rules that might vary from one type of claim to another.

3.25.3. Security Levels

The security level of the non-token subordinate modules should always be less than or equal to that of the containing modules in the case of non-token submodules. It makes no sense for a module of lesser security to be signing claims of a module of higher security. An example of this is a TEE signing claims made by the non-TEE parts (e.g. the high-level OS) of the device.

The opposite may be true for the nested tokens. They usually have their own more secure key material. An example of this is an embedded secure element.

3.25.4. Submodule Names

The label or name for each submodule in the submods map is a text string naming the submodule. No submodules may have the same name.

3.25.5. CDDL for submods

The submodule type is distinguished in the encoded bytes by its data type, map/object for a Claims-Set, string for nested token and array for a detached submodule. Nested tokens are byte-string wrapped when encoded in CBOR and base64 encoded for JSON.

```
$$claims-set-claims // = (submods-label => { + text => Submodule })
```

```
Submodule = Claims-Set / Nested-Token / Detached-Submodule-Digest
```

```
Detached-Submodule-Digest = [  
    algorithm : int / text,  
    digest : bstr  
]
```

4. Unprotected JWT Claims-Sets

This is simply the JSON equivalent of an Unprotected CWT Claims-Set [UCCS.Draft].

It has no protection of its own so protections must be provided by the protocol carrying it. These are extensively discussed in [UCCS.Draft]. All the security discussion and security considerations in [UCCS.Draft] apply to UJCS.

(Note: The EAT author is open to this definition being moved into the UCCS draft, perhaps along with the related CDDL. It is place here for now so that the current UCCS draft plus this document are complete. UJCS is needed for the same use cases that a UCCS is needed. Further, JSON will commonly be used to convey Attestation Results since JSON is common for server to server communications. Server to server communications will often have established security (e.g., TLS) therefore the signing and encryption from JWS and JWE are unnecessary and burdensome).

5. Detached EAT Bundles

A detached EAT bundle is a structure to convey a fully-formed and signed token plus detached claims set that relate to that token. It is a top-level EAT message like a CWT, JWT, UCCS and UJCS. It can be used any place that CWT, JWT, UCCS or UJCS messages are used. It may also be sent as a submodule.

A DEB has two main parts.

The first part is a full top-level token. This top-level token must have at least one submodule that is a detached digest. This top-level token may be either CBOR or JSON-encoded. It may be a CWT, JWT, UCCS or UJCS, but not a DEB. The same mechanism for distinguishing the type for nested token submodules is used here.

The second part is a map/object containing the detached Claims-Sets corresponding to the detached digests in the full token. When the DEB is CBOR-encoded, each Claims-Set is wrapped in a byte string. When the DEB is JSON-encoded, each Claims-Set is base64url encoded. All the detached Claims-Sets MUST be encoded in the same format as the DEB. No mixing of encoding formats is allowed for the Claims-Sets in a DEB.

For CBOR-encoded DEBs, tag TBD602 can be used to identify it. The normal rules apply for use or non-use of a tag. When it is sent as a submodule, it is always sent as a tag to distinguish it from the other types of nested tokens.

The digests of the detached claims sets are associated with detached claims-sets by label/name. It is up to the constructor of the detached EAT bundle to ensure the names uniquely identify the detached claims sets. Since the names are used only in the detached EAT bundle, they can be very short, perhaps one byte.

```
Detached-EAT-Bundle = [  
  main-token : Nested-Token,  
  detached-claims-sets: {  
    + tstr => cbor-wrapped-claims-set / json-wrapped-claims-set  
  }  
]
```

```
json-wrapped-claims-set = tstr .regexp "[A-Za-z0-9_=-]+"
```

```
cbor-wrapped-claims-set = bstr .cbor Claims-Set
```

6. Endorsements and Verification Keys

The Verifier must possess the correct key when it performs the cryptographic part of an EAT verification (e.g., verifying the COSE/JOSE signature). This section describes several ways to identify the verification key. There is not one standard method.

The verification key itself may be a public key, a symmetric key or something complicated in the case of a scheme like Direct Anonymous Attestation (DAA).

RATS Architecture [RATS.Architecture] describes what is called an Endorsement. This is an input to the Verifier that is usually the basis of the trust placed in an EAT and the Attester that generated it. It may contain the public key for verification of the signature on the EAT. It may contain Reference Values to which EAT claims are compared as part of the verification process. It may contain implied claims, those that are passed on to the Relying Party in Attestation Results.

There is not yet any standard format(s) for an Endorsement. One format that may be used for an Endorsement is an X.509 certificate. Endorsement data like Reference Values and implied claims can be carried in X.509 v3 extensions. In this use, the public key in the X.509 certificate becomes the verification key, so identification of the Endorsement is also identification of the verification key.

The verification key identification and establishment of trust in the EAT and the attester may also be by some other means than an Endorsement.

For the components (Attester, Verifier, Relying Party,...) of a particular end-end attestation system to reliably interoperate, its definition should specify how the verification key is identified. Usually, this will be in the profile document for a particular attestation system.

6.1. Identification Methods

Following is a list of possible methods of key identification. A specific attestation system may employ any one of these or one not listed here.

The following assumes Endorsements are X.509 certificates or equivalent and thus does not mention or define any identifier for Endorsements in other formats. If such an Endorsement format is created, new identifiers for them will also need to be created.

6.1.1. COSE/JWS Key ID

The COSE standard header parameter for Key ID (kid) may be used. See [RFC8152] and [RFC7515]

COSE leaves the semantics of the key ID open-ended. It could be a record locator in a database, a hash of a public key, an input to a

KDF, an authority key identifier (AKI) for an X.509 certificate or other. The profile document should specify what the key ID's semantics are.

6.1.2. JWS and COSE X.509 Header Parameters

COSE X.509 [COSE.X509.Draft] and JSON Web Signature [RFC7515] define several header parameters (x5t, x5u,...) for referencing or carrying X.509 certificates any of which may be used.

The X.509 certificate may be an Endorsement and thus carrying additional input to the Verifier. It may be just an X.509 certificate, not an Endorsement. The same header parameters are used in both cases. It is up to the attestation system design and the Verifier to determine which.

6.1.3. CBOR Certificate COSE Header Parameters

Compressed X.509 and CBOR Native certificates are defined by CBOR Certificates [CBOR.Cert.Draft]. These are semantically compatible with X.509 and therefore can be used as an equivalent to X.509 as described above.

These are identified by their own header parameters (c5t, c5u,...).

6.1.4. Claim-Based Key Identification

For some attestation systems, a claim may be re-used as a key identifier. For example, the UEID uniquely identifies the entity and therefore can work well as a key identifier or Endorsement identifier.

This has the advantage that key identification requires no additional bytes in the EAT and makes the EAT smaller.

This has the disadvantage that the unverified EAT must be substantially decoded to obtain the identifier since the identifier is in the COSE/JOSE payload, not in the headers.

6.2. Other Considerations

In all cases there must be some way that the verification key is itself verified or determined to be trustworthy. The key identification itself is never enough. This will always be by some out-of-band mechanism that is not described here. For example, the Verifier may be configured with a root certificate or a master key by the Verifier system administrator.

Often an X.509 certificate or an Endorsement carries more than just the verification key. For example, an X.509 certificate might have key usage constraints and an Endorsement might have Reference Values. When this is the case, the key identifier must be either a protected header or in the payload such that it is cryptographically bound to the EAT. This is in line with the requirements in section 6 on Key Identification in JSON Web Signature [RFC7515].

7. Profiles

This EAT specification does not guarantee that implementations of it will interoperate. The variability in this specification is necessary to accommodate the widely varying use cases. An EAT profile narrows the specification for a specific use case. An ideal EAT profile will guarantee interoperability.

The profile can be named in the token using the profile claim described in Section 3.20.

A profile can apply to Attestation Evidence or to Attestation Results or both.

7.1. Format of a Profile Document

A profile document doesn't have to be in any particular format. It may be simple text, something more formal or a combination.

In some cases CDDL may be created that replaces CDDL in this or other document to express some profile requirements. For example, to require the altitude data item in the location claim, CDDL can be written that replicates the location claim with the altitude no longer optional.

7.2. List of Profile Issues

The following is a list of EAT, CWT, UCCS, JWS, UJCS, COSE, JOSE and CBOR options that a profile should address.

7.2.1. Use of JSON, CBOR or both

The profile should indicate whether the token format should be CBOR, JSON, both or even some other encoding. If some other encoding, a specification for how the CDDL described here is serialized in that encoding is necessary.

This should be addressed for the top-level token and for any nested tokens. For example, a profile might require all nested tokens to be of the same encoding of the top level token.

7.2.2. CBOR Map and Array Encoding

The profile should indicate whether definite-length arrays/maps, indefinite-length arrays/maps or both are allowed. A good default is to allow only definite-length arrays/maps.

An alternate is to allow both definite and indefinite-length arrays/maps. The decoder should accept either. Encoders that need to fit on very small hardware or be actually implement in hardware can use indefinite-length encoding.

This applies to individual EAT claims, CWT and COSE parts of the implementation.

7.2.3. CBOR String Encoding

The profile should indicate whether definite-length strings, indefinite-length strings or both are allowed. A good default is to allow only definite-length strings. As with map and array encoding, allowing indefinite-length strings can be beneficial for some smaller implementations.

7.2.4. CBOR Preferred Serialization

The profile should indicate whether encoders must use preferred serialization. The profile should indicate whether decoders must accept non-preferred serialization.

7.2.5. COSE/JOSE Protection

COSE and JOSE have several options for signed, MACed and encrypted messages. EAT/CWT has the option to have no protection using UCCS and JOSE has a NULL protection option. It is possible to implement no protection, sign only, MAC only, sign then encrypt and so on. All combinations allowed by COSE, JOSE, JWT, CWT, UCCS and UJCS are allowed by EAT.

The profile should list the protections that must be supported by all decoders implementing the profile. The encoders then must implement a subset of what is listed for the decoders, perhaps only one.

Implementations may choose to sign or MAC before encryption so that the implementation layer doing the signing or MACing can be the smallest. It is often easier to make smaller implementations more secure, perhaps even implementing in solely in hardware. The key material for a signature or MAC is a private key, while for encryption it is likely to be a public key. The key for encryption requires less protection.

7.2.6. COSE/JOSE Algorithms

The profile document should list the COSE algorithms that a Verifier must implement. The Attester will select one of them. Since there is no negotiation, the Verifier should implement all algorithms listed in the profile. If detached submodules are used, the COSE algorithms allowed for their digests should also be in the profile.

7.2.7. DEB Support

A Detatched EAT Bundle Section 5 is a special case message that will not often be used. A profile may prohibit its use.

7.2.8. Verification Key Identification

Section Section 6 describes a number of methods for identifying a verification key. The profile document should specify one of these or one that is not described. The ones described in this document are only roughly described. The profile document should go into the full detail.

7.2.9. Endorsement Identification

Similar to, or perhaps the same as Verification Key Identification, the profile may wish to specify how Endorsements are to be identified. However note that Endorsement Identification is optional, where as key identification is not.

7.2.10. Freshness

Just about every use case will require some means of knowing the EAT is recent enough and not a replay of an old token. The profile should describe how freshness is achieved. The section on Freshness in [RATS.Architecture] describes some of the possible solutions to achieve this.

7.2.11. Required Claims

The profile can list claims whose absence results in Verification failure.

7.2.12. Prohibited Claims

The profile can list claims whose presence results in Verification failure.

7.2.13. Additional Claims

The profile may describe entirely new claims. These claims can be required or optional.

7.2.14. Refined Claim Definition

The profile may lock down optional aspects of individual claims. For example, it may require altitude in the location claim, or it may require that HW Versions always be described using EAN-13.

7.2.15. CBOR Tags

The profile should specify whether the token should be a CWT Tag or not. Similarly, the profile should specify whether the token should be a UCCS tag or not.

When COSE protection is used, the profile should specify whether COSE tags are used or not. Note that RFC 8392 requires COSE tags be used in a CWT tag.

Often a tag is unnecessary because the surrounding or carrying protocol identifies the object as an EAT.

7.2.16. Manifests and Software Evidence Claims

The profile should specify which formats are allowed for the manifests and software evidence claims. The profile may also go on to say which parts and options of these formats are used, allowed and prohibited.

8. Encoding and Collected CDDL

An EAT is fundamentally defined using CDDL. This document specifies how to encode the CDDL in CBOR or JSON. Since CBOR can express some things that JSON can't (e.g., tags) or that are expressed differently (e.g., labels) there is some CDDL that is specific to the encoding format.

8.1. Claims-Set and CDDL for CWT and JWT

CDDL was not used to define CWT or JWT. It was not available at the time.

This document defines CDDL for both CWT and JWT as well as UCCS. This document does not change the encoding or semantics of anything in a CWT or JWT.

A Claims-Set is the central data structure for EAT, CWT, JWT and UCCS. It holds all the claims and is the structure that is secured by signing or other means. It is not possible to define EAT, CWT, JWT or UCCS in CDDL without it. The CDDL definition of Claims-Set here is applicable to EAT, CWT, JWT and UCCS.

This document specifies how to encode a Claims-Set in CBOR or JSON.

With the exception of nested tokens and some other externally defined structures (e.g., SWIDs) an entire Claims-Set must be encoded in either CBOR or JSON, never a mixture.

CDDL for the seven claims defined by [RFC8392] and [RFC7519] is included here.

8.2. Encoding Data Types

This makes use of the types defined in [RFC8610] Appendix D, Standard Prelude.

8.2.1. Common Data Types

time-int is identical to the epoch-based time, but disallows floating-point representation.

Unless explicitly indicated, URIs are not the URI tag defined in [RFC8949]. They are just text strings that contain a URI.

string-or-uri = tstr

time-int = #6.1(int)

8.2.2. JSON Interoperability

JSON should be encoded per [RFC8610] Appendix E. In addition, the following CDDL types are encoded in JSON as follows:

- o bstr - must be base64url encoded
- o time - must be encoded as NumericDate as described section 2 of [RFC7519].
- o string-or-uri - must be encoded as StringOrURI as described section 2 of [RFC7519].
- o uri - must be a URI [RFC3986].

- o oid - encoded as a string using the well established dotted-decimal notation (e.g., the text "1.2.250.1").

8.2.3. Labels

Map labels, including Claims-Keys and Claim-Names, and enumerated-type values are always integers when encoding in CBOR and strings when encoding in JSON. There is an exception to this for naming submodules and detached claims sets in a DEB. These are strings in CBOR.

The CDDL in most cases gives both the integer label and the string label as it is not convenient to have conditional CDDL for such.

8.3. CBOR Interoperability

CBOR allows data items to be serialized in more than one form. If the sender uses a form that the receiver can't decode, there will not be interoperability.

This specification gives no blanket requirements to narrow CBOR serialization for all uses of EAT. This allows individual uses to tailor serialization to the environment. It also may result in EAT implementations that don't interoperate.

One way to guarantee interoperability is to clearly specify CBOR serialization in a profile document. See Section 7 for a list of serialization issues that should be addressed.

EAT will be commonly used where the entity generating the attestation is constrained and the receiver/Verifier of the attestation is a capacious server. Following is a set of serialization requirements that work well for that use case and are guaranteed to interoperate. Use of this serialization is recommended where possible, but not required. An EAT profile may just reference the following section rather than spell out serialization details.

8.3.1. EAT Constrained Device Serialization

- o Preferred serialization described in section 4.1 of [RFC8949] is not required. The EAT decoder must accept all forms of number serialization. The EAT encoder may use any form it wishes.
- o The EAT decoder must accept indefinite length arrays and maps as described in section 3.2.2 of [RFC8949]. The EAT encoder may use indefinite length arrays and maps if it wishes.

- o The EAT decoder must accept indefinite length strings as described in section 3.2.3 of [RFC8949]. The EAT encoder may use indefinite length strings if it wishes.
- o Sorting of maps by key is not required. The EAT decoder must not rely on sorting.
- o Deterministic encoding described in Section 4.2 of [RFC8949] is not required.
- o Basic validity described in section 5.3.1 of [RFC8949] must be followed. The EAT encoder must not send duplicate map keys/labels or invalid UTF-8 strings.

8.4. Collected Common CDDL

```

Claims-Set = {
    * $$claims-set-claims,
    * Claim-Label .feature "extended-label" => any
}

Claim-Label = int / text
string-or-uri = tstr

time-int = #6.1(int)
$$claims-set-claims //= (iss-label => text)
$$claims-set-claims //= (sub-label => text)
$$claims-set-claims //= (aud-label => text)
$$claims-set-claims //= (exp-label => ~time)
$$claims-set-claims //= (nbf-label => ~time)
$$claims-set-claims //= (iat-label => ~time)

$$claims-set-claims //=
    (nonce-label => nonce-type / [ 2* nonce-type ])

nonce-type = bstr .size (8..64)
$$claims-set-claims //= (ueid-label => ueid-type)

ueid-type = bstr .size (7..33)
$$claims-set-claims //= (sueids-label => sueids-type)

sueids-type = {
    + tstr => ueid-type
}

oemid-pen = int

oemid-ieee = bstr .size 3

```

```
oemid-random = bstr .size 16

$$claims-set-claims //= (
    oemid-label =>
        oemid-random / oemid-ieee / oemid-pen
)
$$claims-set-claims //= (
    hardware-version-label => hardware-version-type
)

hardware-version-type = [
    version: tstr,
    scheme: $version-scheme
]
hardware-model-type = bytes .size (1..32)

$$claims-set-claims //= (
    hardware-model-label => hardware-model-type
)
$$claims-set-claims //= ( sw-name-label => tstr )
$$claims-set-claims //= (sw-version-label => sw-version-type)

sw-version-type = [
    version: tstr,
    scheme: $version-scheme ; As defined by CoSWID
]
$$claims-set-claims //= (
    security-level-label =>
        security-level-cbor-type /
        security-level-json-type
)

security-level-cbor-type = &(
    unrestricted: 1,
    restricted: 2,
    secure-restricted: 3,
    hardware: 4
)

security-level-json-type =
    "unrestricted" /
    "restricted" /
    "secure-restricted" /
    "hardware"
$$claims-set-claims //= (secure-boot-label => bool)
$$claims-set-claims //= (
    debug-status-label =>
```

```

        debug-status-cbor-type / debug-status-json-type
    )

    debug-status-cbor-type = &(
        enabled: 0,
        disabled: 1,
        disabled-since-boot: 2,
        disabled-permanently: 3,
        disabled-fully-and-permanently: 4
    )

    debug-status-json-type =
        "enabled" /
        "disabled" /
        "disabled-since-boot" /
        "disabled-permanently" /
        "disabled-fully-and-permanently"
    $$claims-set-claims // = (location-label => location-type)

    location-type = {
        latitude => number,
        longitude => number,
        ? altitude => number,
        ? accuracy => number,
        ? altitude-accuracy => number,
        ? heading => number,
        ? speed => number,
        ? timestamp => ~time-int,
        ? age => uint
    }

    latitude = 1 / "latitude"
    longitude = 2 / "longitude"
    altitude = 3 / "altitude"
    accuracy = 4 / "accuracy"
    altitude-accuracy = 5 / "altitude-accuracy"
    heading = 6 / "heading"
    speed = 7 / "speed"
    timestamp = 8 / "timestamp"
    age = 9 / "age"
    $$claims-set-claims // = (uptime-label => uint)
    $$claims-set-claims // = (boot-seed-label => bytes)
    $$claims-set-claims // = (odometer-label => uint)
    $$claims-set-claims // = (
        intended-use-label =>
            intended-use-cbor-type / intended-use-json-type
    )

```

```
intended-use-cbor-type = &(
    generic: 1,
    registration: 2,
    provisioning: 3,
    csr: 4,
    pop: 5
)

intended-use-json-type =
    "generic" /
    "registration" /
    "provisioning" /
    "csr" /
    "pop"
$$claims-set-claims //= (
    dloas-label => [ + dloa-type ]
)

dloa-type = [
    dloa_registrar: ~uri
    dloa_platform_label: text
    ? dloa_application_label: text
]
$$claims-set-claims //= (profile-label => ~uri / ~oid)
$$claims-set-claims //= (
    manifests-label => manifests-type
)

manifests-type = [+ $$manifest-formats]

coswid-that-is-a-cbor-tag-xx = tagged-coswid<concise-swid-tag>

$$manifest-formats /= bytes .cbor coswid-that-is-a-cbor-tag-xx$$claims-set-claims
//= (
    swevidence-label => swevidence-type
)

swevidence-type = [+ $$swevidence-formats]

coswid-that-is-a-cbor-tag = tagged-coswid<concise-swid-tag>
$$swevidence-formats /= bytes .cbor coswid-that-is-a-cbor-tag
$$claims-set-claims //= (swresults-label => [ + swresult-type ])

verification-result-cbor-type = &(
    verification-not-run: 1,
    verification-indeterminate: 2,
    verification-failed: 3,
    fully-verified: 4,
    partially-verified: 5,
```



```
)

verification-result-json-type =
    "verification-not-run" /
    "verification-indeterminate" /
    "verification-failed" /
    "fully-verified" /
    "partially-verified"

verification-objective-cbor-type = &(
    all: 1,
    firmware: 2,
    kernel: 3,
    privileged: 4,
    system-libs: 5,
    partial: 6,
)

verification-objective-json-type =
    "all" /
    "firmware" /
    "kernel" /
    "privileged" /
    "system-libs" /
    "partial"

swresult-type = [
    verification-system: tstr,
    objective: verification-objective-cbor-type /
        verification-objective-json-type,
    result: verification-result-cbor-type /
        verification-result-json-type,
    ? objective-name: tstr
]
$$claims-set-claims // = (submods-label => { + text => Submodule })

Submodule = Claims-Set / Nested-Token / Detached-Submodule-Digest

Detached-Submodule-Digest = [
    algorithm : int / text,
    digest : bstr
]
Detached-EAT-Bundle = [
    main-token : Nested-Token,
    detached-claims-sets: {
        + tstr => cbor-wrapped-claims-set / json-wrapped-claims-set
    }
]
```

```
    }  
  ]
```

```
json-wrapped-claims-set = tstr .regexp "[A-Za-z0-9_=-]+"
```

```
cbor-wrapped-claims-set = bstr .cbor Claims-Set
```

8.5. Collected CDDL for CBOR

```
CBOR-Token = Tagged-CBOR-Token / Untagged-CBOR-Token
```

```
Tagged-CBOR-Token  = CWT-Tagged-Message  
Tagged-CBOR-Token /= UCCS-Tagged-Message  
Tagged-CBOR-Token /= DEB-Tagged-Message
```

```
Untagged-CBOR-Token  = CWT-Untagged-Message  
Untagged-CBOR-Token /= UCCS-Untagged-Message  
Untagged-CBOR-Token /= DEB-Untagged-Message
```

```
CWT-Tagged-Message = COSE_Tagged_Message  
CWT-Untagged-Message = COSE_Untagged_Message
```

```
UCCS-Message = UCCS-Tagged-Message / UCCS-Untagged-Message
```

```
UCCS-Tagged-Message = #6.601(UCCS-Untagged-Message)
```

```
UCCS-Untagged-Message = Claims-Set
```

```
DEB-Tagged-Message = #6.602(DEB-Untagged-Message)
```

```
DEB-Untagged-Message = Detached-EAT-Bundle
```

```
Nested-Token =  
  tstr / ; A JSON-encoded Nested-Token (see json-nested-token.cddl)  
  bstr .cbor Tagged-CBOR-Token
```

```
iss-label = 1  
sub-label = 2  
aud-label = 3
```

```

exp-label = 4
nbf-label = 5
iat-label = 6
cti-label = 7nonce-label = 10
ueid-label = 256
sueids-label = 257
oemid-label = 258
hardware-model-label = 259
hardware-version-label = 260
secure-boot-label = 262
debug-status-label = 263
location-label = 264
profile-label = 265
submods-label = 266
security-level-label = <TBD>
uptime-label = <TBD>
boot-seed-label = <TB>
odometer-label = <TBD>
intended-use-label = <TBD>
dloas-label = <TBD>
sw-name-label = <TBD>
sw-version-label = <TBD>
manifests-label = <TBD>
swevidence-label = <TBD>
swresults-label = <TBD>

```

8.6. Collected CDDL for JSON

```
JWT-Message = text .regexp [A-Za-z0-9_=-]+\.[A-Za-z0-9_=-]+\.[A-Za-z0-9_=-]+
```

```
UJCS-Message = Claims-Set
```

```

Nested-Token = [
  type : "JWT" / "CBOR" / "UJCS" / "DEB",
  nested-token : JWT-Message /
                  B64URL-Tagged-CBOR-Token /
                  DEB-JSON-Message /
                  UJCS-Message
]

```

```

B64URL-Tagged-CBOR-Token = tstr .regexp "[A-Za-z0-9_=-]+"
iss-label = "iss"
sub-label = "sub"
aud-label = "aud"

```

```
exp-label = "exp"
nbf-label = "nbf"
iat-label = "iat"
cti-label = "cti"nonce-label /= "nonce"
```

```
ueid-label /= "ueid"
sueids-label /= "sueids"
oemid-label /= "oemid"
hardware-model-label /= "hwmodel"
hardware-version-label /= "hwversion"
security-level-label /= "seclevel"
secure-boot-label /= "secboot"
debug-status-label /= "dbgstat"
location-label /= "location"
profile-label /= "eat-profile"
uptime-label /= "uptime"
boot-seed-label /= "bootseed"
odometer-label /= "odometer"
intended-use-label /= "intuse"
dloas-label /= "dloas"
sw-name-label /= "swname"
sw-version-label /= "swversion"
manifests-label /= "manifests"
swevidence-label /= "swevidence"
swresults-label /= "swresults"
submods-label /= "submods"
```

```
latitude /= "lat"
longitude /= "long"
altitude /= "alt"
accuracy /= "accry"
altitude-accuracy /= "alt-accry"
heading /= "heading"
speed /= "speed"
```

9. IANA Considerations

9.1. Reuse of CBOR and JSON Web Token (CWT and JWT) Claims Registries

Claims defined for EAT are compatible with those of CWT and JWT so the CWT and JWT Claims Registries, [IANA.CWT.Claims] and [IANA.JWT.Claims], are re used. No new IANA registry is created.

All EAT claims defined in this document are placed in both registries. All new EAT claims defined subsequently should be placed in both registries.

9.2. Claim Characteristics

The following is design guidance for creating new EAT claims, particularly those to be registered with IANA.

Much of this guidance is generic and could also be considered when designing new CWT or JWT claims.

9.2.1. Interoperability and Relying Party Orientation

It is a broad goal that EATs can be processed by Relying Parties in a general way regardless of the type, manufacturer or technology of the device from which they originate. It is a goal that there be general-purpose verification implementations that can verify tokens for large numbers of use cases with special cases and configurations for different device types. This is a goal of interoperability of the semantics of claims themselves, not just of the signing, encoding and serialization formats.

This is a lofty goal and difficult to achieve broadly requiring careful definition of claims in a technology neutral way. Sometimes it will be difficult to design a claim that can represent the semantics of data from very different device types. However, the goal remains even when difficult.

9.2.2. Operating System and Technology Neutral

Claims should be defined such that they are not specific to an operating system. They should be applicable to multiple large high-level operating systems from different vendors. They should also be applicable to multiple small embedded operating systems from multiple vendors and everything in between.

Claims should not be defined such that they are specific to a SW environment or programming language.

Claims should not be defined such that they are specific to a chip or particular hardware. For example, they should not just be the contents of some HW status register as it is unlikely that the same HW status register with the same bits exists on a chip of a different manufacturer.

The boot and debug state claims in this document are an example of a claim that has been defined in this neutral way.

9.2.3. Security Level Neutral

Many use cases will have EATs generated by some of the most secure hardware and software that exists. Secure Elements and smart cards are examples of this. However, EAT is intended for use in low-security use cases the same as high-security use case. For example, an app on a mobile device may generate EATs on its own.

Claims should be defined and registered on the basis of whether they are useful and interoperable, not based on security level. In particular, there should be no exclusion of claims because they are just used only in low-security environments.

9.2.4. Reuse of Extant Data Formats

Where possible, claims should use already standardized data items, identifiers and formats. This takes advantage of the expertise put into creating those formats and improves interoperability.

Often extant claims will not be defined in an encoding or serialization format used by EAT. It is preferred to define a CBOR and JSON format for them so that EAT implementations do not require a plethora of encoders and decoders for serialization formats.

In some cases, it may be better to use the encoding and serialization as is. For example, signed X.509 certificates and CRLs can be carried as-is in a byte string. This retains interoperability with the extensive infrastructure for creating and processing X.509 certificates and CRLs.

9.2.5. Proprietary Claims

EAT allows the definition and use of proprietary claims.

For example, a device manufacturer may generate a token with proprietary claims intended only for verification by a service offered by that device manufacturer. This is a supported use case.

In many cases proprietary claims will be the easiest and most obvious way to proceed, however for better interoperability, use of general standardized claims is preferred.

9.3. Claims Registered by This Document

This specification adds the following values to the "JSON Web Token Claims" registry established by [RFC7519] and the "CBOR Web Token Claims Registry" established by [RFC8392]. Each entry below is an

addition to both registries (except for the nonce claim which is already registered for JWT, but not registered for CWT).

The "Claim Description", "Change Controller" and "Specification Documents" are common and equivalent for the JWT and CWT registries. The "Claim Key" and "Claim Value Types(s)" are for the CWT registry only. The "Claim Name" is as defined for the CWT registry, not the JWT registry. The "JWT Claim Name" is equivalent to the "Claim Name" in the JWT registry.

9.3.1. Claims for Early Assignment

RFC Editor: in the final publication this section should be combined with the following section as it will no longer be necessary to distinguish claims with early assignment. Also, the following paragraph should be removed.

The claims in this section have been (requested for / given) early assignment according to [RFC7120]. They have been assigned values and registered before final publication of this document. While their semantics is not expected to change in final publication, it is possible that they will. The JWT Claim Names and CWT Claim Keys are not expected to change.

In draft -06 an early allocation was described. The processing of that early allocation was never correctly completed. This early allocation assigns different numbers for the CBOR claim labels. This early allocation will presumably complete correctly

- o Claim Name: Nonce
- o Claim Description: Nonce
- o JWT Claim Name: "nonce" (already registered for JWT)
- o Claim Key: TBD (requested value 10)
- o Claim Value Type(s): byte string
- o Change Controller: IESG
- o Specification Document(s): [OpenIDConnectCore], *this document*
- o Claim Name: UEID
- o Claim Description: The Universal Entity ID
- o JWT Claim Name: "ueid"

- o CWT Claim Key: TBD (requested value 256)
- o Claim Value Type(s): byte string
- o Change Controller: IESG
- o Specification Document(s): *this document*
- o Claim Name: SUEIDs
- o Claim Description: Semi-permanent UEIDs
- o JWT Claim Name: "sueids"
- o CWT Claim Key: TBD (requested value 257)
- o Claim Value Type(s): map
- o Change Controller: IESG
- o Specification Document(s): *this document*
- o Claim Name: Hardware OEMID
- o Claim Description: Hardware OEM ID
- o JWT Claim Name: "oemid"
- o Claim Key: TBD (requested value 258)
- o Claim Value Type(s): byte string or integer
- o Change Controller: IESG
- o Specification Document(s): *this document*
- o Claim Name: Hardware Model
- o Claim Description: Model identifier for hardware
- o JWT Claim Name: "hwmodel"
- o Claim Key: TBD (requested value 259)
- o Claim Value Type(s): byte string
- o Change Controller: IESG

- o Specification Document(s): *this document*
- o Claim Name: Hardware Version
- o Claim Description: Hardware Version Identifier
- o JWT Claim Name: "hwversion"
- o Claim Key: TBD (requested value 260)
- o Claim Value Type(s): array
- o Change Controller: IESG
- o Specification Document(s): *this document*
- o Claim Name: Secure Boot
- o Claim Description: Indicate whether the boot was secure
- o JWT Claim Name: "secboot"
- o Claim Key: TBD (requested value 262)
- o Claim Value Type(s): Boolean
- o Change Controller: IESG
- o Specification Document(s): *this document*
- o Claim Name: Debug Status
- o Claim Description: Indicate status of debug facilities
- o JWT Claim Name: "dbgstat"
- o Claim Key: TBD (requested value 263)
- o Claim Value Type(s): integer or string
- o Change Controller: IESG
- o Specification Document(s): *this document*
- o Claim Name: Location
- o Claim Description: The geographic location

- o JWT Claim Name: "location"
- o Claim Key: TBD (requested value 264)
- o Claim Value Type(s): map
- o Change Controller: IESG
- o Specification Document(s): *this document*
- o Claim Name: Profile
- o Claim Description: Indicates the EAT profile followed
- o JWT Claim Name: "eat_profile"
- o Claim Key: TBD (requested value 265)
- o Claim Value Type(s): URI or OID
- o Change Controller: IESG
- o Specification Document(s): *this document*
- o Claim Name: Submodules Section
- o Claim Description: The section containing submodules
- o JWT Claim Name: "submods"
- o Claim Key: TBD (requested value 266)
- o Claim Value Type(s): map
- o Change Controller: IESG
- o Specification Document(s): *this document*

9.3.2. To be Assigned Claims

(Early assignment is NOT requested for these claims. Implementers should be aware they may change)

- o Claim Name: Security Level
- o Claim Description: Characterization of the security of an Attester or submodule

- o JWT Claim Name: "secclevel"
- o Claim Key: TBD
- o Claim Value Type(s): integer or string
- o Change Controller: IESG
- o Specification Document(s): *this document*
- o Claim Name: Uptime
- o Claim Description: Uptime
- o JWT Claim Name: "uptime"
- o Claim Key: TBD
- o Claim Value Type(s): unsigned integer
- o Change Controller: IESG
- o Specification Document(s): *this document*
- o Claim Name: Boot Seed
- o Claim Description: Identifies a boot cycle
- o JWT Claim Name: "bootseed"
- o Claim Key: TBD
- o Claim Value Type(s): bytes
- o Change Controller: IESG
- o Specification Document(s): *this document*
- o Claim Name: Intended Use
- o Claim Description: Indicates intended use of the EAT
- o JWT Claim Name: "intuse"
- o Claim Key: TBD
- o Claim Value Type(s): integer or string

- o Change Controller: IESG
- o Specification Document(s): *this document*
- o Claim Name: DLOAs
- o Claim Description: Certifications received as Digital Letters of Approval
- o JWT Claim Name: "dloas"
- o Claim Key: TBD
- o Claim Value Type(s): array
- o Change Controller: IESG
- o Specification Document(s): *this document*
- o Claim Name: SW Name
- o Claim Description: The name of the SW running in the entity
- o JWT Claim Name: "swname"
- o Claim Key: TBD
- o Claim Value Type(s): map
- o Change Controller: IESG
- o Specification Document(s): *this document*
- o Claim Name: SW Version
- o Claim Description: The version of SW running in the entity
- o JWT Claim Name: "swversion"
- o Claim Key: TBD
- o Claim Value Type(s): map
- o Change Controller: IESG
- o Specification Document(s): *this document*
- o Claim Name: SW Manifests

- o Claim Description: Manifests describing the SW installed on the entity
- o JWT Claim Name: "manifests"
- o Claim Key: TBD
- o Claim Value Type(s): array
- o Change Controller: IESG
- o Specification Document(s): *this document*
- o Claim Name: SW Evidence
- o Claim Description: Measurements of the SW, memory configuration and such on the entity
- o JWT Claim Name: "swevidence"
- o Claim Key: TBD
- o Claim Value Type(s): array
- o Change Controller: IESG
- o Specification Document(s): *this document*
- o Claim Name: SW Measurment Results
- o Claim Description: The results of comparing SW measurements to reference values
- o JWT Claim Name: "swresults"
- o Claim Key: TBD
- o Claim Value Type(s): array
- o Change Controller: IESG
- o Specification Document(s): *this document*

9.3.3. Version Schemes Registered by this Document

IANA is requested to register a new value in the "Software Tag Version Scheme Values" established by [CoSWID].

The new value is a version scheme a 13-digit European Article Number [EAN-13]. An EAN-13 is also known as an International Article Number or most commonly as a bar code. This version scheme is the ASCII text representation of EAN-13 digits, the same ones often printed with a bar code. This version scheme must comply with the EAN allocation and assignment rules. For example, this requires the manufacturer to obtain a manufacture code from GS1.

Index	Version Scheme Name	Specification
5	ean-13	This document

9.3.4. UEID URN Registered by this Document

IANA is requested to register the following new subtypes in the "DEV URN Subtypes" registry under "Device Identification". See [RFC9039].

Subtype	Description	Reference
ueid	Universal Entity Identifier	This document
sueid	Semi-permanent Universal Entity Identifier	This document

9.3.5. Tag for Detached EAT Bundle

In the registry [IANA.cbor-tags], IANA is requested to allocate the following tag from the FCFS space, with the present document as the specification reference.

Tag	Data Items	Semantics
TBD602	array	Detached EAT Bundle Section 5

10. Privacy Considerations

Certain EAT claims can be used to track the owner of an entity and therefore, implementations should consider providing privacy-preserving options dependent on the intended usage of the EAT. Examples would include suppression of location claims for EAT's provided to unauthenticated consumers.

10.1. UEID and SUEID Privacy Considerations

A UEID is usually not privacy-preserving. Any set of Relying Parties that receives tokens that happen to be from a particular entity will be able to know the tokens are all from the same entity and be able to track it.

Thus, in many usage situations UEID violates governmental privacy regulation. In other usage situations a UEID will not be allowed for certain products like browsers that give privacy for the end user. It will often be the case that tokens will not have a UEID for these reasons.

An SUEID is also usually not privacy-preserving. In some cases it may have fewer privacy issues than a UEID depending on when and how and when it is generated.

There are several strategies that can be used to still be able to put UEIDs and SUEIDs in tokens:

- o The entity obtains explicit permission from the user of the entity to use the UEID/SUEID. This may be through a prompt. It may also be through a license agreement. For example, agreements for some online banking and brokerage services might already cover use of a UEID/SUEID.
- o The UEID/SUEID is used only in a particular context or particular use case. It is used only by one Relying Party.
- o The entity authenticates the Relying Party and generates a derived UEID/SUEID just for that particular Relying Party. For example, the Relying Party could prove their identity cryptographically to the entity, then the entity generates a UEID just for that Relying Party by hashing a proofed Relying Party ID with the main entity UEID/SUEID.

Note that some of these privacy preservation strategies result in multiple UEIDs and SUEIDs per entity. Each UEID/SUEID is used in a different context, use case or system on the entity. However, from the view of the Relying Party, there is just one UEID and it is still globally universal across manufacturers.

10.2. Location Privacy Considerations

Geographic location is most always considered personally identifiable information. Implementers should consider laws and regulations governing the transmission of location data from end user devices to servers and services. Implementers should consider using location

management facilities offered by the operating system on the entity generating the attestation. For example, many mobile phones prompt the user for permission when before sending location data.

10.3. Replay Protection and Privacy

EAT offers 2 primary mechanisms for token replay protection (also sometimes known as token "freshness"): the cti/jti claim and the nonce claim. The cti/jti claim in a CWT/JWT is a field that may be optionally included in the EAT and is in general derived on the same device in which the entity is instantiated. The nonce claim is based on a value that is usually derived remotely (outside of the entity). These claims can be used to extract and convey personally-identifying information either inadvertently or by intention. For instance, an implementor may choose a cti that is equivalent to a username associated with the device (e.g., account login). If the token is inspected by a 3rd-party then this information could be used to identify the source of the token or an account associated with the token (e.g., if the account name is used to derive the nonce). In order to avoid the conveyance of privacy-related information in either the cti/jti or nonce claims, these fields should be derived using a salt that originates from a true and reliable random number generator or any other source of randomness that would still meet the target system requirements for replay protection.

11. Security Considerations

The security considerations provided in Section 8 of [RFC8392] and Section 11 of [RFC7519] apply to EAT in its CWT and JWT form, respectively. In addition, implementors should consider the following.

11.1. Key Provisioning

Private key material can be used to sign and/or encrypt the EAT, or can be used to derive the keys used for signing and/or encryption. In some instances, the manufacturer of the entity may create the key material separately and provision the key material in the entity itself. The manufacturer of any entity that is capable of producing an EAT should take care to ensure that any private key material be suitably protected prior to provisioning the key material in the entity itself. This can require creation of key material in an enclave (see [RFC4949] for definition of "enclave"), secure transmission of the key material from the enclave to the entity using an appropriate protocol, and persistence of the private key material in some form of secure storage to which (preferably) only the entity has access.

11.1.1. Transmission of Key Material

Regarding transmission of key material from the enclave to the entity, the key material may pass through one or more intermediaries. Therefore some form of protection ("key wrapping") may be necessary. The transmission itself may be performed electronically, but can also be done by human courier. In the latter case, there should be minimal to no exposure of the key material to the human (e.g. encrypted portable memory). Moreover, the human should transport the key material directly from the secure enclave where it was created to a destination secure enclave where it can be provisioned.

11.2. Transport Security

As stated in Section 8 of [RFC8392], "The security of the CWT relies upon on the protections offered by COSE". Similar considerations apply to EAT when sent as a CWT. However, EAT introduces the concept of a nonce to protect against replay. Since an EAT may be created by an entity that may not support the same type of transport security as the consumer of the EAT, intermediaries may be required to bridge communications between the entity and consumer. As a result, it is RECOMMENDED that both the consumer create a nonce, and the entity leverage the nonce along with COSE mechanisms for encryption and/or signing to create the EAT.

Similar considerations apply to the use of EAT as a JWT. Although the security of a JWT leverages the JSON Web Encryption (JWE) and JSON Web Signature (JWS) specifications, it is still recommended to make use of the EAT nonce.

11.3. Multiple EAT Consumers

In many cases, more than one EAT consumer may be required to fully verify the entity attestation. Examples include individual consumers for nested EATs, or consumers for individual claims with an EAT. When multiple consumers are required for verification of an EAT, it is important to minimize information exposure to each consumer. In addition, the communication between multiple consumers should be secure.

For instance, consider the example of an encrypted and signed EAT with multiple claims. A consumer may receive the EAT (denoted as the "receiving consumer"), decrypt its payload, verify its signature, but then pass specific subsets of claims to other consumers for evaluation ("downstream consumers"). Since any COSE encryption will be removed by the receiving consumer, the communication of claim subsets to any downstream consumer should leverage a secure protocol (e.g. one that uses transport-layer security, i.e. TLS),

However, assume the EAT of the previous example is hierarchical and each claim subset for a downstream consumer is created in the form of a nested EAT. Then transport security between the receiving and downstream consumers is not strictly required. Nevertheless, downstream consumers of a nested EAT should provide a nonce unique to the EAT they are consuming.

12. References

12.1. Normative References

- [CoSWID] Birkholz, H., Fitzgerald-McKay, J., Schmidt, C., and D. Waltermire, "Concise Software Identification Tags", draft-ietf-sacm-coswid-20 (work in progress), January 2022.
- [DLOA] "Digital Letter of Approval", November 2015, <https://globalplatform.org/wp-content/uploads/2015/12/GPC_DigitalLetterOfApproval_v1.0.pdf>.
- [EAN-13] GS1, "International Article Number - EAN/UPC barcodes", 2019, <<https://www.gs1.org/standards/barcodes/ean-upc>>.
- [FIDO.AROE] The FIDO Alliance, "FIDO Authenticator Allowed Restricted Operating Environments List", November 2020, <<https://fidoalliance.org/specs/fido-security-requirements/fido-authenticator-allowed-restricted-operating-environments-list-v1.2-fd-20201102.html>>.
- [IANA.cbor-tags] "IANA CBOR Tags Registry", n.d., <<https://www.iana.org/assignments/cbor-tags/cbor-tags.xhtml>>.
- [IANA.CWT.Claims] IANA, "CBOR Web Token (CWT) Claims", <<http://www.iana.org/assignments/cwt>>.
- [IANA.JWT.Claims] IANA, "JSON Web Token (JWT) Claims", <<https://www.iana.org/assignments/jwt>>.
- [OpenIDConnectCore] Sakimura, N., Bradley, J., Jones, M., Medeiros, B. D., and C. Mortimore, "OpenID Connect Core 1.0 incorporating errata set 1", November 2014, <https://openid.net/specs/openid-connect-core-1_0.html>.

- [PEN] "Private Enterprise Number (PEN) Request", n.d.,
<<https://pen.iana.org/pen/PenApplication.page>>.
- [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>>.
- [RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <<https://www.rfc-editor.org/info/rfc3986>>.
- [RFC7159] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", RFC 7159, DOI 10.17487/RFC7159, March 2014, <<https://www.rfc-editor.org/info/rfc7159>>.
- [RFC7515] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Signature (JWS)", RFC 7515, DOI 10.17487/RFC7515, May 2015, <<https://www.rfc-editor.org/info/rfc7515>>.
- [RFC7516] Jones, M. and J. Hildebrand, "JSON Web Encryption (JWE)", RFC 7516, DOI 10.17487/RFC7516, May 2015, <<https://www.rfc-editor.org/info/rfc7516>>.
- [RFC7517] Jones, M., "JSON Web Key (JWK)", RFC 7517, DOI 10.17487/RFC7517, May 2015, <<https://www.rfc-editor.org/info/rfc7517>>.
- [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/info/rfc7519>>.
- [RFC7800] Jones, M., Bradley, J., and H. Tschofenig, "Proof-of-Possession Key Semantics for JSON Web Tokens (JWTs)", RFC 7800, DOI 10.17487/RFC7800, April 2016, <<https://www.rfc-editor.org/info/rfc7800>>.
- [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/info/rfc8126>>.
- [RFC8152] Schaad, J., "CBOR Object Signing and Encryption (COSE)", RFC 8152, DOI 10.17487/RFC8152, July 2017, <<https://www.rfc-editor.org/info/rfc8152>>.

- [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/info/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/info/rfc8392>>.
- [RFC8610] Birkholz, H., Vigano, C., and C. Bormann, "Concise Data Definition Language (CDDL): A Notational Convention to Express Concise Binary Object Representation (CBOR) and JSON Data Structures", RFC 8610, DOI 10.17487/RFC8610, June 2019, <<https://www.rfc-editor.org/info/rfc8610>>.
- [RFC8747] Jones, M., Seitz, L., Selander, G., Erdtman, S., and H. Tschofenig, "Proof-of-Possession Key Semantics for CBOR Web Tokens (CWTs)", RFC 8747, DOI 10.17487/RFC8747, March 2020, <<https://www.rfc-editor.org/info/rfc8747>>.
- [RFC8949] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", STD 94, RFC 8949, DOI 10.17487/RFC8949, December 2020, <<https://www.rfc-editor.org/info/rfc8949>>.
- [RFC9090] Bormann, C., "Concise Binary Object Representation (CBOR) Tags for Object Identifiers", RFC 9090, DOI 10.17487/RFC9090, July 2021, <<https://www.rfc-editor.org/info/rfc9090>>.
- [ThreeGPP.IMEI]
3GPP, "3rd Generation Partnership Project; Technical Specification Group Core Network and Terminals; Numbering, addressing and identification", 2019, <<https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=729>>.
- [UCCS.Draft]
Birkholz, H., O'Donoghue, J., Cam-Winget, N., and C. Bormann, "A CBOR Tag for Unprotected CWT Claims Sets", draft-ietf-rats-uccs-02 (work in progress), January 2022.
- [WGS84] National Geospatial-Intelligence Agency (NGA), "WORLD GEODETIC SYSTEM 1984, NGA.STND.0036_1.0.0_WGS84", July 2014, <<https://earth-info.nga.mil/php/download.php?file=coord-wgs84>>.

12.2. Informative References

- [BirthdayAttack]
"Birthday attack",
<https://en.wikipedia.org/wiki/Birthday_attack>.
- [CBOR.Cert.Draft]
Mattsson, J. P., Selander, G., Raza, S., Hoeglund, J., and
M. Furuheid, "CBOR Encoded X.509 Certificates (C509
Certificates)", draft-ietf-cose-chor-encoded-cert-03 (work
in progress), January 2022.
- [Common.Criteria]
"Common Criteria for Information Technology Security
Evaluation", April 2017,
<<https://www.commoncriteriaportal.org/cc/>>.
- [COSE.X509.Draft]
Schaad, J., "CBOR Object Signing and Encryption (COSE):
Header parameters for carrying and referencing X.509
certificates", draft-ietf-cose-x509-08 (work in progress),
December 2020.
- [FIPS-140]
National Institute of Standards, "Security Requirements for
Cryptographic Modules", May 2001,
<<https://csrc.nist.gov/publications/detail/fips/140/2/final>>.
- [IEEE.802-2001]
"IEEE Standard For Local And Metropolitan Area Networks
Overview And Architecture", 2007,
<<https://webstore.ansi.org/standards/ieee/ieee8022001r2007>>.
- [IEEE.802.1AR]
"IEEE Standard, "IEEE 802.1AR Secure Device Identifier",
December 2009, <<http://standards.ieee.org/findstds/standard/802.1AR-2009.html>>.
- [IEEE.RA]
"IEEE Registration Authority",
<<https://standards.ieee.org/products-services/regauth/index.html>>.

[OUI.Guide]

"Guidelines for Use of Extended Unique Identifier (EUI), Organizationally Unique Identifier (OUI), and Company ID (CID)", August 2017,
<<https://standards.ieee.org/content/dam/ieee-standards/standards/web/documents/tutorials/eui.pdf>>.

[OUI.Lookup]

"IEEE Registration Authority Assignments",
<<https://regauth.standards.ieee.org/standards-ra-web/pub/view.html#registries>>.

[RATS.Architecture]

Birkholz, H., Thaler, D., Richardson, M., Smith, N., and W. Pan, "Remote Attestation Procedures Architecture", draft-ietf-rats-architecture-15 (work in progress), February 2022.

[RFC4122] Leach, P., Mealling, M., and R. Salz, "A Universally Unique IDentifier (UUID) URN Namespace", RFC 4122, DOI 10.17487/RFC4122, July 2005,
<<https://www.rfc-editor.org/info/rfc4122>>.

[RFC4422] Melnikov, A., Ed. and K. Zeilenga, Ed., "Simple Authentication and Security Layer (SASL)", RFC 4422, DOI 10.17487/RFC4422, June 2006,
<<https://www.rfc-editor.org/info/rfc4422>>.

[RFC4949] Shirey, R., "Internet Security Glossary, Version 2", FYI 36, RFC 4949, DOI 10.17487/RFC4949, August 2007,
<<https://www.rfc-editor.org/info/rfc4949>>.

[RFC7120] Cotton, M., "Early IANA Allocation of Standards Track Code Points", BCP 100, RFC 7120, DOI 10.17487/RFC7120, January 2014, <<https://www.rfc-editor.org/info/rfc7120>>.

[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>>.

[RFC9039] Arkko, J., Jennings, C., and Z. Shelby, "Uniform Resource Names for Device Identifiers", RFC 9039, DOI 10.17487/RFC9039, June 2021,
<<https://www.rfc-editor.org/info/rfc9039>>.

[W3C.GeoLoc]

Worldwide Web Consortium, "Geolocation API Specification
2nd Edition", January 2018, <[https://www.w3.org/TR/
geolocation-API/#coordinates_interface](https://www.w3.org/TR/geolocation-API/#coordinates_interface)>.

Appendix A. Examples

These examples are either UCCS, shown as CBOR diagnostic, or UJCS messages. Full CWT and JWT examples with signing and encryption are not given.

All UCCS examples can be the payload of a CWT. To do so, they must be converted from the UCCS message to a Claims-Set, which is achieved by "removing" the tag.

UJCS messages can be directly used as the payload of a JWT.

WARNING: These examples use tag and label numbers not yet assigned by IANA.

A.1. Simple TEE Attestation

This is a simple attestation of a TEE that includes a manifest that is a payload CoSWID to describe the TEE's software.

/ This is a UCCS EAT that describes a simple TEE. /

```
601({
  / nonce /          10: h'948f8860d13a463e',
  / security-level / 261: 3, / secure-restricted /
  / secure-boot /    262: true,
  / debug-status /   263: 2, / disabled-since-boot /
  / manifests /      273: [
                                / This is byte-string wrapped /
                                / payload CoSWID. It gives the TEE /
                                / software name, the version and /
                                / the name of the file it is in. /
                                h' da53574944a60064336132340c01016b
                                41636d6520544545204f530d65332e31
                                2e340282a2181f6b41636d6520544545
                                204f53182101a2181f6b41636d652054
                                4545204f5318210206a111a118186e61
                                636d655f7465655f332e657865'
                                ]
  })
```


/ A payload CoSWID created by the SW vendor. All this really does /
 / is name the TEE SW, its version and lists the one file that /
 / makes up the TEE. /

```
1398229316({
  / Unique CoSWID ID /      0: "3a24",
  / tag-version /          12: 1,
  / software-name /        1: "Acme TEE OS",
  / software-version /     13: "3.1.4",
  / entity /               2: [
                                {
                                  / entity-name /      31: "Acme TEE OS",
                                  / role /              33: 1 / tag-creator /
                                },
                                {
                                  / entity-name /      31: "Acme TEE OS",
                                  / role /              33: 2 / software-creator /
                                }
                              ],
  / payload /              6: {
    / ...file /            17: {
      / ...fs-name /       24: "acme_tee_3.exe"
    }
  }
})
```

A.2. Submodules for Board and Device

```

/ This example shows use of submodules to give information /
/ about the chip, board and overall device. /
/
/ The main attestation is associated with the chip with the /
/ CPU and running the main OS. It is what has the keys and /
/ produces the token. /
/
/ The board is made by a different vendor than the chip. /
/ Perhaps it is some generic IoT board. /
/
/ The device is some specific appliance that is made by a /
/ different vendor than either the chip or the board. /
/
/ Here the board and device submodules aren't the typical /
/ target environments as described by the RATS architecture /
/ document, but they are a valid use of submodules. /

{
  / nonce /          10: h'948f8860d13a463e8e',
  / UEID /           256: h'0198f50a4ff6c05861c8860d13a638ea',
  / HW OEM ID /      258: h'894823', / IEEE OUI format OEM ID /
  / HW Model ID /    259: h'549dcecc8b987c737b44e40f7c635ce8'
                        / Hash of chip model name /,
  / HW Version /     260: ["1.3.4", 1], / Multipartnumeric version /
  / SW Name /        271: "Acme OS",
  / SW Version /     272: ["3.5.5", 1],
  / secure-boot /    262: true,
  / debug-status /   263: 3, / permanent-disable /
  / timestamp (iat) / 6: 1526542894,
  / security-level / 261: 3, / secure restricted OS /
  / submods / 266: {
    / A submodule to hold some claims about the circuit board /
    "board" : {
      / HW OEM ID /    258: h'9bef8787ebal3e2c8f6e7cb4blf4619a',
      / HW Model ID / 259: h'ee80f5a66c1fb9742999a8fdab930893'
                        / Hash of board module name /,
      / HW Version /   260: ["2.0a", 2] / multipartnumeric+suffix /
    },

    / A submodule to hold claims about the overall device /
    "device" : {
      / HW OEM ID /    258: 61234, / PEN Format OEM ID /
      / HW Version /   260: ["4012345123456", 5] / EAN-13 format (barcode) /
    }
  }
}

```

A.3. EAT Produced by Attestation Hardware Block

```

/ This is an example of a token produced by a HW block      /
/ purpose-built for attestation. Only the nonce claim changes /
/ from one attestation to the next as the rest either come   /
/ directly from the hardware or from one-time-programmable memory /
/ (e.g. a fuse). 47 bytes encoded in CBOR (8 byte nonce, 16 byte /
/ UEID). /

601({
  / nonce /          10: h'948f8860d13a463e',
  / UEID /           256: h'0198f50a4ff6c05861c8860d13a638ea',
  / OEMID /          258: 64242, / Private Enterprise Number /
  / security-level / 261: 4, / hardware level security /
  / secure-boot /    262: true,
  / debug-status /   263: 3, / disabled-permanently /
  / HW version /     260: [ "3.1", 1 ] / Type is multipartnumeric /
})

```

A.4. Detached EAT Bundle

In this DEB main token is produced by a HW attestation block. The detached Claims-Set is produced by a TEE and is largely identical to the Simple TEE examples above. The TEE digests its Claims-Set and feeds that digest to the HW block.

In a better example the attestation produced by the HW block would be a CWT and thus signed and secured by the HW block. Since the signature covers the digest from the TEE that Claims-Set is also secured.

The DEB itself can be assembled by untrusted SW.

```

/ This is a detached EAT bundle (DEB) tag.  /

602([

  / First part is a full EAT token with claims like nonce and /
  / UEID. Most importantly, it includes a submodule that is a /
  / detached digest which is the hash of the "TEE" claims set /
  / in the next section.                                     /
  /                                                         /
  / This token here is in UCCS format (unsigned). In a more /
  / realistic example, it would be a signed CWT.           /
  h'd90259a80a48948f8860d13a463e190100500198
  f50a4ff6c05861c8860d13a638ea19010219faf2
  19010504190106f5190107031901048263332e31
  0119010aa163544545822f5820e5cf95fd24fab7
  1446742dd58d43dae178e55fe2b94291a9291082
  ffc2635a0b',
  {
    / A CBOR-encoded byte-string wrapped EAT claims-set. It /
    / contains claims suitable for a TEE                       /
    "TEE" : h'a50a48948f8860d13a463e19010503190106f519
           01070219011181585dda53574944a60064336132
           340c01016b41636d6520544545204f530d65332e
           312e340282a2181f6b41636d6520544545204f53
           182101a2181f6b41636d6520544545204f531821
           0206a111a118186e61636d655f7465655f332e65
           7865'
  }
])

```

```

/ This example contains submodule that is a detached digest, /
/ which is the hash of a Claims-Set convey outside this token. /
/ Other than that is is the other example of a token from an /
/ attestation HW block /

601({
  / nonce /          10: h'948f8860d13a463e',
  / UEID /           256: h'0198f50a4ff6c05861c8860d13a638ea',
  / OEMID /          258: 64242, / Private Enterprise Number /
  / security-level / 261: 4, / hardware level security /
  / secure-boot /    262: true,
  / debug-status /   263: 3, / disabled-permanently /
  / hw version /     260: [ "3.1", 1 ], / multipartnumeric /
  / submods/         266: {
                                "TEE": [ / detached digest submod /
                                          -16, / SHA-256 /
                                          h'e5cf95fd24fab7144674
                                          2dd58d43dae178e55fe2
                                          b94291a9291082ffc2635
                                          a0b'
                                ]
  }
})

```

A.5. Key / Key Store Attestation

```

/ This is an attestation of a public key and the key store /
/ implementation that protects and manages it. The key store /
/ implementation is in a security-oriented execution /
/ environment separate from the high-level OS, for example a /
/ TEE. The key store is the Attester. /
/ /
/ There is some attestation of the high-level OS, just version /
/ and boot & debug status. It is a Claims-Set submodule because /
/ it has lower security level than the key store. The key /
/ store's implementation has access to info about the HLOS, so /
/ it is able to include it. /
/ /
/ A key and an indication of the user authentication given to /
/ allow access to the key is given. The labels for these are /
/ in the private space since this is just a hypothetical /
/ example, not part of a standard protocol. /
/ /
/ This is similar to Android Key Attestation. /

```

```
601({
```

```

/ nonce /          10: h'948f8860d13a463e',
/ security-level / 261: 3, / secure-restricted /
/ secure-boot /    262: true,
/ debug-status /   263: 2, / disabled-since-boot /
/ manifests /      273: [
                                h'da53574944a600683762623334383766
                                0c000169436172626f6e6974650d6331
                                2e320e0102a2181f75496e6475737472
                                69616c204175746f6d6174696f6e1821
                                02'
                                / Above is an encoded CoSWID      /
                                / with the following data          /
                                /   SW Name: "Carbonite"            /
                                /   SW Vers: "1.2"                  /
                                /   SW Creator:                      /
                                /     "Industrial Automation"       /
                                ],
/ expiration /      4: 1634324274, / 2021-10-15T18:57:54Z /
/ creation time /   6: 1634317080, / 2021-10-15T16:58:00Z /
-80000 : "fingerprint",
-80001 : { / The key -- A COSE_Key /
/ kty /          1: 2, / EC2, elliptic curve with x & y /
/ kid /          2: h'36675c206f96236c3f51f54637b94ced',
/ curve /        -1: 2, / curve is P-256 /
/ x-coord /       -2: h'65eda5a12577c2bae829437fe338701a
                    10aaa375e1bb5b5de108de439c08551d',
/ y-coord /       -3: h'1e52ed75701163f7f9e40ddf9f341b3d
                    c9ba860af7e0ca7ca7e9eecd0084d19c'
},

/ submods /        266 : {
                    "HLOS" : { / submod for high-level OS /
/ nonce /          10: h'948f8860d13a463e',
/ security-level / 261: 1, / unrestricted /
/ secure-boot /    262: true,
/ manifests /      273: [
                                h'da53574944a600687337
                                6537346b78380c000168
                                44726f6964204f530d65
                                52322e44320e0302a218
                                1f75496e647573747269
                                616c204175746f6d6174
                                696f6e182102'
                                / Above is an encoded CoSWID /
                                / with the following data:      /
                                /   SW Name: "Droid OS"         /
                                /   SW Vers: "R2.D2"            /
                                /   SW Creator:                  /

```

```
        /      "Industrial Automation"/  
    ]  
    }  
}  
))
```

A.6. SW Measurements of an IoT Device

This is a simple token that might be for an IoT device. It includes CoSWID format measurements of the SW. The CoSWID is in byte-string wrapped in the token and also shown in diagnostic form.

```

/ This EAT UCCS is for an IoT device with a TEE. The attestation /
/ is produced by the TEE. There is a submodule for the IoT OS (the /
/ main OS of the IoT device that is not as secure as the TEE). The /
/ submodule contains claims for the IoT OS. The TEE also measures /
/ the IoT OS and puts the measurements in the submodule. /

```

```

601({
  / nonce /          10: h'948f8860d13a463e',
  / security-level / 261: 3, / secure-restricted /
  / secure-boot /    262: true,
  / debug-status /   263: 2, / disabled-since-boot /
  / OEMID /          258: h'8945ad', / IEEE CID based /
  / UEID /           256: h'0198f50a4ff6c05861c8860d13a638ea',
  / sumods /         266: {
    "OS" : {
      / security-level / 261: 2, / restricted /
      / secure-boot /    262: true,
      / debug-status /   263: 2, / disabled-since-boot /
      / swevidence /     274: [
        / This is a byte-string wrapped /
        / evidence CoSWID. It has /
        / hashes of the main files of /
        / the IoT OS. /
        h'da53574944a600663463613234350c
        17016d41636d6520522d496f542d4f
        530d65332e312e3402a2181f724163
        6d6520426173652041747465737465
        7218210103a11183a318187161636d
        655f725f696f745f6f732e65786514
        1a0044b349078201582005f6b327c1
        73b4192bd2c3ec248a292215eab456
        611bf7a783e25c1782479905a31818
        6d7265736f75726365732e72736314
        1a000c38b10782015820c142b9aba4
        280c4bb8c75f716a43c99526694caa
        be529571f5569bb7dc542f98a31818
        6a636f6d6d6f6e2e6c6962141a0023
        3d3b0782015820a6a9dcdfb3884da5
        f884e4e1e8e8629958c2dbc7027414
        43a913e34de9333be6'
      ]
    }
  }
})

```

```

/ An evidence CoSWID created for the "Acme R-IoT-OS" created by /
/ the "Acme Base Attester" (both fictitious names). It provides /

```



```

/ measurements of the SW (other than the attester SW) on the /
/ device. /

1398229316({
  / Unique CoSWID ID /      0: "4ca245",
  / tag-version /          12: 23, / Attester-maintained counter /
  / software-name /        1: "Acme R-IoT-OS",
  / software-version /     13: "3.1.4",
  / entity /               2: {
    / entity-name /        31: "Acme Base Attester",
    / role /               33: 1 / tag-creator /
  },
  / evidence /             3: {
    / ...file /            17: [
      {
        / ...fs-name /     24: "acme_r_iot_os.exe",
        / ...size /        20: 4502345,
        / ...hash /        7: [
          1, / SHA-256 /
          h'05f6b327c173b419
            2bd2c3ec248a2922
            15eab456611bf7a7
            83e25c1782479905'
        ]
      },
      {
        / ...fs-name /     24: "resources.rsc",
        / ...size /        20: 800945,
        / ...hash /        7: [
          1, / SHA-256 /
          h'c142b9aba4280c4b
            b8c75f716a43c995
            26694caabe529571
            f5569bb7dc542f98'
        ]
      },
      {
        / ...fs-name /     24: "common.lib",
        / ...size /        20: 2309435,
        / ...hash /        7: [
          1, / SHA-256 /
          h'a6a9dcdfb3884da5
            f884e4e1e8e86299
            58c2dbc702741443
            a913e34de9333be6'
        ]
      }
    ]
  }
}
]

```

```
    }  
  })
```

A.7. Attestation Results in JSON format

This is a UJCS format token that might be the output of a Verifier that evaluated the IoT Attestation example immediately above.

This particular Verifier knows enough about the TEE Attester to be able to pass claims like security level directly through to the Relying Party. The Verifier also knows the Reference Values for the measured SW components and is able to check them. It informs the Relying Party that they were correct in the swresults claim. "Trustus Verifications" is the name of the services that verifies the SW component measurements.

This UJCS is identical to JSON-encoded Claims-Set that could be a JWT payload.

```
{  
  "nonce" : "lI+IYNE6Rj4=",  
  "secllevel" : "secure-restricted",  
  "secboot" : true,  
  "dbgstat" : "disabled-since-boot",  
  "OEMID" : "iUWt",  
  "UEID" : "AZjlCk/2wFhhyIYNE6Y4",  
  "submods" : {  
    "secllevel" : "restricted",  
    "secboot" : true,  
    "dbgstat" : "disabled-since-boot",  
    "swname" : "Acme R-IoT-OS",  
    "sw-version" : [  
      "3.1.4"  
    ],  
    "swresults" : [  
      [  
        "Trustus Verifications",  
        "all",  
        "fully-verified"  
      ]  
    ]  
  }  
}
```

Appendix B. UEID Design Rationale

B.1. Collision Probability

This calculation is to determine the probability of a collision of UEIDs given the total possible entity population and the number of entities in a particular entity management database.

Three different sized databases are considered. The number of devices per person roughly models non-personal devices such as traffic lights, devices in stores they shop in, facilities they work in and so on, even considering individual light bulbs. A device may have individually attested subsystems, for example parts of a car or a mobile phone. It is assumed that the largest database will have at most 10% of the world's population of devices. Note that databases that handle more than a trillion records exist today.

The trillion-record database size models an easy-to-imagine reality over the next decades. The quadrillion-record database is roughly at the limit of what is imaginable and should probably be accommodated. The 100 quadrillion database is highly speculative perhaps involving nanorobots for every person, livestock animal and domesticated bird. It is included to round out the analysis.

Note that the items counted here certainly do not have IP address and are not individually connected to the network. They may be connected to internal buses, via serial links, Bluetooth and so on. This is not the same problem as sizing IP addresses.

People	Devices / Person	Subsystems / Device	Database Portion	Database Size
10 billion	100	10	10%	trillion (10 ¹²)
10 billion	100,000	10	10%	quadrillion (10 ¹⁵)
100 billion	1,000,000	10	10%	100 quadrillion (10 ¹⁷)

This is conceptually similar to the Birthday Problem where m is the number of possible birthdays, always 365, and k is the number of people. It is also conceptually similar to the Birthday Attack where collisions of the output of hash functions are considered.

The proper formula for the collision calculation is

$$p = 1 - e^{\{-k^2/(2n)\}}$$

p Collision Probability
 n Total possible population
 k Actual population

However, for the very large values involved here, this formula requires floating point precision higher than commonly available in calculators and SW so this simple approximation is used. See [BirthdayAttack].

$$p = k^2 / 2n$$

For this calculation:

p Collision Probability
 n Total population based on number of bits in UEID
 k Population in a database

Database Size	128-bit UEID	192-bit UEID	256-bit UEID
trillion (10 ¹²)	2 * 10 ⁻¹⁵	8 * 10 ⁻³⁵	5 * 10 ⁻⁵⁵
quadrillion (10 ¹⁵)	2 * 10 ⁻⁰⁹	8 * 10 ⁻²⁹	5 * 10 ⁻⁴⁹
100 quadrillion (10 ¹⁷)	2 * 10 ⁻⁰⁵	8 * 10 ⁻²⁵	5 * 10 ⁻⁴⁵

Next, to calculate the probability of a collision occurring in one year's operation of a database, it is assumed that the database size is in a steady state and that 10% of the database changes per year. For example, a trillion record database would have 100 billion states per year. Each of those states has the above calculated probability of a collision.

This assumption is a worst-case since it assumes that each state of the database is completely independent from the previous state. In reality this is unlikely as state changes will be the addition or deletion of a few records.

The following tables gives the time interval until there is a probability of a collision based on there being one tenth the number of states per year as the number of records in the database.

$$t = 1 / ((k / 10) * p)$$

t Time until a collision
 p Collision probability for UEID size
 k Database size

Database Size	128-bit UEID	192-bit UEID	256-bit UEID
trillion (10 ¹²)	60,000 years	10 ²⁴ years	10 ⁴⁴ years
quadrillion (10 ¹⁵)	8 seconds	10 ¹⁴ years	10 ³⁴ years
100 quadrillion (10 ¹⁷)	8 microseconds	10 ¹¹ years	10 ³¹ years

Clearly, 128 bits is enough for the near future thus the requirement that UEIDs be a minimum of 128 bits.

There is no requirement for 256 bits today as quadrillion-record databases are not expected in the near future and because this time-to-collision calculation is a very worst case. A future update of the standard may increase the requirement to 256 bits, so there is a requirement that implementations be able to receive 256-bit UEIDs.

B.2. No Use of UUID

A UEID is not a UUID [RFC4122] by conscious choice for the following reasons.

UUIDs are limited to 128 bits which may not be enough for some future use cases.

Today, cryptographic-quality random numbers are available from common CPUs and hardware. This hardware was introduced between 2010 and 2015. Operating systems and cryptographic libraries give access to this hardware. Consequently, there is little need for implementations to construct such random values from multiple sources on their own.

Version 4 UUIDs do allow for use of such cryptographic-quality random numbers, but do so by mapping into the overall UUID structure of time and clock values. This structure is of no value here yet adds complexity. It also slightly reduces the number of actual bits with entropy.

UUIDs seem to have been designed for scenarios where the implementor does not have full control over the environment and uniqueness has to be constructed from identifiers at hand. UEID takes the view that

hardware, software and/or manufacturing process directly implement UEID in a simple and direct way. It takes the view that cryptographic quality random number generators are readily available as they are implemented in commonly used CPU hardware.

Appendix C. EAT Relation to IEEE.802.1AR Secure Device Identity (DevID)

This section describes several distinct ways in which an IEEE IDevID [IEEE.802.1AR] relates to EAT, particularly to UEID and SUEID.

[IEEE.802.1AR] orients around the definition of an implementation called a "DevID Module." It describes how IDevIDs and LDevIDs are stored, protected and accessed using a DevID Module. A particular level of defense against attack that should be achieved to be a DevID is defined. The intent is that IDevIDs and LDevIDs are used with an open set of network protocols for authentication and such. In these protocols the DevID secret is used to sign a nonce or similar to proof the association of the DevID certificates with the device.

By contrast, EAT defines network protocol for proving trustworthiness to a Relying Party, the very thing that is not defined in [IEEE.802.1AR]. Nor does not give details on how keys, data and such are stored protected and accessed. EAT is intended to work with a variety of different on-device implementations ranging from minimal protection of assets to the highest levels of asset protection. It does not define any particular level of defense against attack, instead providing a set of security considerations.

EAT and DevID can be viewed as complimentary when used together or as competing to provide a device identity service.

C.1. DevID Used With EAT

As just described, EAT defines a network protocol and [IEEE.802.1AR] doesn't. Vice versa, EAT doesn't define a device implementation and DevID does.

Hence, EAT can be the network protocol that a DevID is used with. The DevID secret becomes the attestation key used to sign EATs. The DevID and its certificate chain become the Endorsement sent to the Verifier.

In this case the EAT and the DevID are likely to both provide a device identifier (e.g. a serial number). In the EAT it is the UEID (or SUEID). In the DevID (used as an endorsement), it is a device serial number included in the subject field of the DevID certificate. It is probably a good idea in this use for them to be the same serial number or for the UEID to be a hash of the DevID serial number.

C.2. How EAT Provides an Equivalent Secure Device Identity

The UEID, SUEID and other claims like OEM ID are equivalent to the secure device identity put into the subject field of a DevID certificate. These EAT claims can represent all the same fields and values that can be put in a DevID certificate subject. EAT explicitly and carefully defines a variety of useful claims.

EAT secures the conveyance of these claims by having them signed on the device by the attestation key when the EAT is generated. EAT also signs the nonce that gives freshness at this time. Since these claims are signed for every EAT generated, they can include things that vary over time like GPS location.

DevID secures the device identity fields by having them signed by the manufacturer of the device sign them into a certificate. That certificate is created once during the manufacturing of the device and never changes so the fields cannot change.

So in one case the signing of the identity happens on the device and the other in a manufacturing facility, but in both cases the signing of the nonce that proves the binding to the actual device happens on the device.

While EAT does not specify how the signing keys, signature process and storage of the identity values should be secured against attack, an EAT implementation may have equal defenses against attack. One reason EAT uses CBOR is because it is simple enough that a basic EAT implementation can be constructed entirely in hardware. This allows EAT to be implemented with the strongest defenses possible.

C.3. An X.509 Format EAT

It is possible to define a way to encode EAT claims in an X.509 certificate. For example, the EAT claims might be mapped to X.509 v3 extensions. It is even possible to stuff a whole CBOR-encoded unsigned EAT token into a X.509 certificate.

If that X.509 certificate is an IDevID or LDevID, this becomes another way to use EAT and DevID together.

Note that the DevID must still be used with an authentication protocol that has a nonce or equivalent. The EAT here is not being used as the protocol to interact with the rely party.

C.4. Device Identifier Permanence

In terms of permanence, an IDevID is similar to a UEID in that they do not change over the life of the device. They cease to exist only when the device is destroyed.

An SUEID is similar to an LDevID. They change on device life-cycle events.

[IEEE.802.1AR] describes much of this permanence as resistant to attacks that seek to change the ID. IDevID permanence can be described this way because [IEEE.802.1AR] is oriented around the definition of an implementation with a particular level of defense against attack.

EAT is not defined around a particular implementation and must work on a range of devices that have a range of defenses against attack. EAT thus can't be defined permanence in terms of defense against attack. EAT's definition of permanence is in terms of operations and device lifecycle.

Appendix D. Changes from Previous Drafts

The following is a list of known changes from the previous drafts. This list is non-authoritative. It is meant to help reviewers see the significant differences.

D.1. From draft-rats-eat-01

- o Added UEID design rationale appendix

D.2. From draft-mandyam-rats-eat-00

This is a fairly large change in the orientation of the document, but no new claims have been added.

- o Separate information and data model using CDDL.
- o Say an EAT is a CWT or JWT
- o Use a map to structure the boot_state and location claims

D.3. From draft-ietf-rats-eat-01

- o Clarifications and corrections for OEMID claim
- o Minor spelling and other fixes

- o Add the nonce claim, clarify jti claim
- D.4. From draft-ietf-rats-eat-02
- o Roll all EUIs back into one UEID type
 - o UEIDs can be one of three lengths, 128, 192 and 256.
 - o Added appendix justifying UEID design and size.
 - o Submods part now includes nested eat tokens so they can be named and there can be more than one of them
 - o Lots of fixes to the CDDL
 - o Added security considerations
- D.5. From draft-ietf-rats-eat-03
- o Split boot_state into secure-boot and debug-disable claims
 - o Debug disable is an enumerated type rather than Booleans
- D.6. From draft-ietf-rats-eat-04
- o Change IMEI-based UEIDs to be encoded as a 14-byte string
 - o CDDL cleaned up some more
 - o CDDL allows for JWTs and UCCSs
 - o CWT format submodules are byte string wrapped
 - o Allows for JWT nested in CWT and vice versa
 - o Allows UCCS (unsigned CWTs) and JWT unsecured tokens
 - o Clarify tag usage when nesting tokens
 - o Add section on key inclusion
 - o Add hardware version claims
 - o Collected CDDL is now filled in. Other CDDL corrections.
 - o Rename debug-disable to debug-status; clarify that it is not extensible

- o Security level claim is not extensible
 - o Improve specification of location claim and added a location privacy section
 - o Add intended use claim
- D.7. From draft-ietf-rats-eat-05
- o CDDL format issues resolved
 - o Corrected reference to Location Privacy section
- D.8. From draft-ietf-rats-eat-06
- o Added boot-seed claim
 - o Rework CBOR interoperability section
 - o Added profiles claim and section
- D.9. From draft-ietf-rats-eat-07
- o Filled in IANA and other sections for possible preassignment of Claim Keys for well understood claims
- D.10. From draft-ietf-rats-eat-08
- o Change profile claim to be either a URL or an OID rather than a test string
- D.11. From draft-ietf-rats-eat-09
- o Add SUEIDs
 - o Add appendix comparing IDevID to EAT
 - o Added section on use for Evidence and Attestation Results
 - o Fill in the key ID and endorsements identificaiton section
 - o Remove origination claim as it is replaced by key IDs and endorsements
 - o Added manifests and software evidence claims
 - o Add string labels non-claim labels for use with JSON (e.g. labels for members of location claim)

- o EAN-13 HW versions are no longer a separate claim. Now they are folded in as a CoSWID version scheme.

D.12. From draft-ietf-rats-eat-10

- o Hardware version is made into an array of two rather than two claims
- o Corrections and wording improvements for security levels claim
- o Add swresults claim
- o Add dloas claim - Digital Letter of Approvals, a list of certifications
- o CDDL for each claim no longer in a separate sub section
- o Consistent use of terminology from RATS architecture document
- o Consistent use of terminology from CWT and JWT documents
- o Remove operating model and procedures; refer to CWT, JWT and RATS architecture instead
- o Some reorganization of Section 1
- o Moved a few references, including RATS Architecture, to informative.
- o Add detached submodule digests and detached eat bundles (DEBs)
- o New simpler and more universal scheme for identifying the encoding of a nested token
- o Made clear that CBOR and JSON are only mixed when nesting a token in another token
- o Clearly separate CDDL for JSON and CBOR-specific data items
- o Define UJCS (unsigned JWTs)
- o Add CDDL for a general Claims-Set used by UCCS, UJCS, CWT, JWT and EAT
- o Top level CDDL for CWT correctly refers to COSE
- o OEM ID is specifically for HW, not for SW

- o HW OEM ID can now be a PEN
- o HW OEM ID can now be a 128-bit random number
- o Expand the examples section
- o Add software and version claims as easy / JSON alternative to CoSWID

D.13. From draft-ietf-rats-eat-11

- o Add HW model claim
- o Change reference for CBOR OID draft to RFC 9090
- o Correct the iat claim in some examples
- o Make HW Version just one claim rather than 3 (device, board and chip)
- o Remove CDDL comments from CDDL blocks
- o More clearly define "entity" and use it more broadly, particularly instead of "device"
- o Re do early allocation of CBOR labels since last one didn't complete correctly
- o Lots of rewording and tightening up of section 1
- o Lots of wording improvements in section 3, particularly better use of normative language
- o Improve wording in submodules section, particularly how to distinguish types when decoding
- o Remove security-level from early allocation
- o Add boot odometer claim
- o Add privacy considerations for replay protection

Authors' Addresses

Laurence Lundblade
Security Theory LLC

EMail: lg1@securitytheory.com

Giridhar Mandyam
Qualcomm Technologies Inc.
5775 Morehouse Drive
San Diego, California
USA

Phone: +1 858 651 7200
EMail: mandyam@qti.qualcomm.com

Jeremy O'Donoghue
Qualcomm Technologies Inc.
279 Farnborough Road
Farnborough GU14 7LS
United Kingdom

Phone: +44 1252 363189
EMail: jodonogh@qti.qualcomm.com

RATS Working Group
Internet-Draft
Intended status: Standards Track
Expires: 17 October 2022

H. Birkholz
M. Eckel
Fraunhofer SIT
S. Bhandari
ThoughtSpot
E. Voit
B. Sulzen
Cisco
L. Xia
Huawei
T. Laffey
HPE
G. Fedorkow
Juniper
15 April 2022

A YANG Data Model for Challenge-Response-based Remote Attestation
Procedures using TPMs
draft-ietf-rats-yang-tpm-charra-19

Abstract

This document defines YANG RPCs and a few configuration nodes required to retrieve attestation evidence about integrity measurements from a device, following the operational context defined in TPM-based Network Device Remote Integrity Verification. Complementary measurement logs are also provided by the YANG RPCs, originating from one or more roots of trust for measurement (RTMs). The module defined requires at least one TPM 1.2 or TPM 2.0 as well as a corresponding TPM Software Stack (TSS), or equivalent hardware implementations that include the protected capabilities as provided by TPMs as well as a corresponding software stack, included in the device components of the composite device the YANG server is running on.

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 17 October 2022.

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
1.1. Requirements notation	3
2. The YANG Module for Basic Remote Attestation Procedures	3
2.1. YANG Modules	3
2.1.1. 'ietf-tpm-remote-attestation'	4
2.1.2. 'ietf-tcg-algs'	33
3. IANA Considerations	48
4. Security Considerations	49
5. References	51
5.1. Normative References	51
5.2. Informative References	56
Appendix A. Integrity Measurement Architecture (IMA)	56
Appendix B. IMA for Network Equipment Boot Logs	57
Authors' Addresses	58

1. Introduction

This document is based on the general terminology defined in the [I-D.ietf-rats-architecture] and uses the operational context defined in [I-D.ietf-rats-tpm-based-network-device-attest] as well as the interaction model and information elements defined in [I-D.ietf-rats-reference-interaction-models]. The currently supported hardware security modules (HSMs) are the Trusted Platform Modules (TPMs) [TPM1.2] and [TPM2.0] as specified by the Trusted Computing Group (TCG). One TPM, or multiple TPMs in the case of a

Composite Device, are required in order to use the YANG module defined in this document. Each TPM is used as a root of trust for storage (RTS) in order to store system security measurement Evidence. And each TPM is used as a root of trust for reporting (RTR) in order to retrieve attestation Evidence. This is done by using a YANG RPC to request a quote which exposes a rolling hash of the security measurements held internally within the TPM.

Specific terms imported from [I-D.ietf-rats-architecture] and used in this document include: Attester, Composite Device, Evidence.

Specific terms imported from [TPM2.0-Key] and used in this document include: Endorsement Key (EK), Initial Attestation Key (IAK), Attestation Identity Key (AIK), Local Attestation Key (LAK).

1.1. 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. The YANG Module for Basic Remote Attestation Procedures

One or more TPMs MUST be embedded in a Composite Device that provides attestation evidence via the YANG module defined in this document. The ietf-tpm-remote-attestation YANG module enables a composite device to take on the role of an Attester, in accordance with the Remote Attestation Procedures (RATS) architecture [I-D.ietf-rats-architecture], and the corresponding challenge-response interaction model defined in the [I-D.ietf-rats-reference-interaction-models] document. A fresh nonce with an appropriate amount of entropy [NIST-915121] MUST be supplied by the YANG client in order to enable a proof-of-freshness with respect to the attestation Evidence provided by the Attester running the YANG datastore. Further, this nonce is used to prevent replay attacks. The method for communicating the relationship of each individual TPM to specific measured component within the Composite Device is out of the scope of this document.

2.1. YANG Modules

In this section the several YANG modules are defined.

2.1.1.1. 'ietf-tpm-remote-attestation'

This YANG module imports modules from [RFC6991] with prefix 'yang', [RFC8348] with prefix 'hw', [I-D.ietf-netconf-keystore] with prefix 'ks', and 'ietf-tcg-algs.yang' Section 2.1.2.3 with prefix 'taa'. Additionally, references are made to [RFC8032], [RFC8017], [RFC6933], [TPM1.2-Commands], [TPM2.0-Arch], [TPM2.0-Structures], [TPM2.0-Key], [TPM1.2-Structures], [bios-log], [BIOS-Log-Event-Type], as well as Appendix A and Appendix B.

2.1.1.1.1. Features

This module supports the following features:

- * 'mtpm': Indicates that multiple TPMs on the device can support remote attestation. For example, this feature could be used in cases where multiple line cards are present, each with its own TPM.
- * 'bios': Indicates that the device supports the retrieval of BIOS/UEFI event logs. [bios-log]
- * 'ima': Indicates that the device supports the retrieval of event logs from the Linux Integrity Measurement Architecture (IMA, see Appendix A).
- * 'netequip_boot': Indicates that the device supports the retrieval of netequip boot event logs. See Appendix A and Appendix B.

2.1.1.1.2. Identities

This module supports the following types of attestation event logs: 'bios', 'ima', and 'netequip_boot'.

2.1.1.1.3. Remote Procedure Calls (RPCs)

In the following, RPCs for both TPM 1.2 and TPM 2.0 attestation procedures are defined.

2.1.1.1.3.1. 'tpm12-challenge-response-attestation'

This RPC allows a Verifier to request signed TPM PCRs (_TPM Quote_ operation) from a TPM 1.2 compliant cryptoprocessor. Where the feature 'mtpm' is active, and one or more 'certificate-name' is not provided, all TPM 1.2 compliant cryptoprocessors will respond. A YANG tree diagram of this RPC is as follows:

```

+---x tpm12-challenge-response-attestation {taa:tpm12}?
+---w input
|   +---w tpm12-attestation-challenge
|       +---w pcr-index*          pcr
|       +---w nonce-value        binary
|       +---w certificate-name*   certificate-name-ref
|                               {tpm:mtpm}?
+---ro output
+---ro tpm12-attestation-response* []
+---ro certificate-name            certificate-name-ref
+---ro up-time?                   uint32
+---ro TPM_QUOTE2?                binary

```

2.1.1.3.2. 'tpm20-challenge-response-attestation'

This RPC allows a Verifier to request signed TPM PCRs (`_TPM Quote_` operation) from a TPM 2.0 compliant cryptoprocessor. Where the feature 'mtpm' is active, and one or more 'certificate-name' is not provided, all TPM 2.0 compliant cryptoprocessors will respond. A YANG tree diagram of this RPC is as follows:

```

+---x tpm20-challenge-response-attestation {taa:tpm20}?
+---w input
|   +---w tpm20-attestation-challenge
|       +---w nonce-value          binary
|       +---w tpm20-pcr-selection* []
|           +---w tpm20-hash-algo? identityref
|           +---w pcr-index*       pcr
|       +---w certificate-name*    certificate-name-ref
|                               {tpm:mtpm}?
+---ro output
+---ro tpm20-attestation-response* []
+---ro certificate-name            certificate-name-ref
+---ro TPMS_QUOTE_INFO            binary
+---ro quote-signature?          binary
+---ro up-time?                   uint32
+---ro unsigned-pcr-values* []
|   +---ro tpm20-hash-algo?       identityref
|   +---ro pcr-values* [pcr-index]
|       +---ro pcr-index          pcr
|       +---ro pcr-value?        binary

```

An example of an RPC challenge requesting PCRs 0-7 from a SHA-256 bank could look like the following:

```
<rpc message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <tpm20-challenge-response-attestation>
    xmlns="urn:ietf:params:xml:ns:yang:ietf-tpm-remote-attestation"
    <certificate-name>
      (identifier of a TPM signature key with which the Verifier is
      supposed to sign the attestation data)
    </certificate-name>
    <nonce>
      0xe041307208d9f78f5b1bbecd19e2d152ad49de2fc5a7d8dbf769f6b8ffdeab9
    </nonce>
    <tpm20-pcr-selection>
      <tpm20-hash-algo
        xmlns="urn:ietf:params:xml:ns:yang:ietf-tcg-algs">
          TPM_ALG_SHA256
        </tpm20-hash-algo>
      <pcr-index>0</pcr-index>
      <pcr-index>1</pcr-index>
      <pcr-index>2</pcr-index>
      <pcr-index>3</pcr-index>
      <pcr-index>4</pcr-index>
      <pcr-index>5</pcr-index>
      <pcr-index>6</pcr-index>
      <pcr-index>7</pcr-index>
    </tpm20-pcr-selection>
  </tpm20-challenge-response-attestation>
</rpc>
```

A successful response could be formatted as follows:

```
<rpc-reply message-id="101"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <tpm20-attestation-response
    xmlns="urn:ietf:params:xml:ns:yang:ietf-tpm-remote-attestation">
    <certificate-name
      xmlns="urn:ietf:params:xml:ns:yang:ietf-keystore">
      (instance of Certificate name in the Keystore)
    </certificate-name>
    <attestation-data>
      (raw attestation data, i.e. the TPM quote; this includes
      a composite digest of requested PCRs, the nonce,
      and TPM 2.0 time information.)
    </attestation-data>
    <quote-signature>
      (signature over attestation-data using the TPM key
      identified by sig-key-id)
    </quote-signature>
  </tpm20-attestation-response>
</rpc-reply>
```

2.1.1.4. 'log-retrieval'

This RPC allows a Verifier to acquire the evidence which was extended into specific TPM PCRs. A YANG tree diagram of this RPC is as follows:

```

+---x log-retrieval
  +---w input
    +---w log-type          identityref
    +---w log-selector* []
      +---w name*           string
      +---w (index-type)?
        +---:(last-entry)
          +---w last-entry-value?  binary
        +---:(index)
          +---w last-index-number?  uint64
        +---:(timestamp)
          +---w timestamp?          yang:date-and-time
    +---w log-entry-quantity?      uint16
  +---ro output
    +---ro system-event-logs
      +---ro node-data* []
        +---ro name?          string
        +---ro up-time?       uint32
        +---ro log-result
          +---ro (attested_event_log_type)
            +---:(bios) {bios}?
              +---ro bios-event-logs
                +---ro bios-event-entry* [event-number]
                  +---ro event-number      uint32
                  +---ro event-type?       uint32
                  +---ro pcr-index?        pcr
                  +---ro digest-list* []
                    +---ro hash-algo?      identityref
                    +---ro digest*         binary
                  +---ro event-size?       uint32
                  +---ro event-data*       binary
            +---:(ima) {ima}?
              +---ro ima-event-logs
                +---ro ima-event-entry* [event-number]
                  +---ro event-number      uint64
                  +---ro ima-template?     string
                  +---ro filename-hint?    string
                  +---ro filedata-hash?    binary
                  +---ro filedata-hash-algorithm? string
                  +---ro template-hash-algorithm? string
                  +---ro template-hash?    binary
                  +---ro pcr-index?        pcr

```

```

|          +---ro signature?                               binary
+---:(netequip_boot) {netequip_boot}?
  +---ro boot-event-logs
    +---ro boot-event-entry* [event-number]
      +---ro event-number                                uint64
      +---ro ima-template?                               string
      +---ro filename-hint?                              string
      +---ro filedata-hash?                              binary
      +---ro filedata-hash-algorithm?                    string
      +---ro template-hash-algorithm?                    string
      +---ro template-hash?                              binary
      +---ro pcr-index?                                  pcr
      +---ro signature?                                  binary

```

2.1.1.5. Data Nodes

This section provides a high level description of the data nodes containing the configuration and operational objects with the YANG model. For more details, please see the YANG model itself in Figure 1.

Container 'rats-support-structures': This houses the set of information relating to remote attestation for a device. This includes specific device TPM(s), the compute nodes (such as line cards) on which the TPM(s) reside, and the algorithms supported across the platform.

Container 'tpms': Provides configuration and operational details for each supported TPM, including the tpm-firmware-version, PCRs which may be quoted, certificates which are associated with that TPM, and the current operational status. Of note are the certificates which are associated with that TPM. As a certificate is associated with a particular TPM attestation key, knowledge of the certificate allows a specific TPM to be identified.

```

+--rw tpms
  +--rw tpm* [name]
    +--rw name string
    +--ro hardware-based boolean
    +--ro physical-index? int32 {hw:entity-mib}?
    +--ro path? string
    +--ro compute-node compute-node-ref {tpm:mtpm}?
    +--ro manufacturer? string
    +--rw firmware-version identityref
    +--rw tpm12-hash-algo? identityref
    +--rw tpm12-pcrs* pcr
    +--rw tpm20-pcr-bank* [tpm20-hash-algo]
      | +--rw tpm20-hash-algo identityref
      | +--rw pcr-index* tpm:pcr
    +--ro status enumeration
    +--rw certificates
      +--rw certificate* [name]
        +--rw name string
        +--rw keystore-ref? leafref {ks:asymmetric-keys}?
        +--rw type? enumeration

```

container 'attester-supported-algos' - Identifies which TCG hash algorithms are available for use on the Attesting platform. An operator will use this information to limit algorithms available for use by RPCs to just a desired set from the universe of all allowed hash algorithms by the TCG.

```

+--rw attester-supported-algos
  +--rw tpm12-asymmetric-signing* identityref
  +--rw tpm12-hash* identityref
  +--rw tpm20-asymmetric-signing* identityref
  +--rw tpm20-hash* identityref

```

container 'compute-nodes' - When there is more than one TPM supported, this container maintains the set of information related to the compute node associated with a specific TPM. This allows each specific TPM to identify to which 'compute-node' it belongs.

```

+--rw compute-nodes {tpm:mtpm}?
  +--ro compute-node* [node-id]
    +--ro node-id string
    +--ro node-physical-index? int32 {hw:entity-mib}?
    +--ro node-name? string
    +--ro node-location? string

```

2.1.1.6. YANG Module

```
<CODE BEGINS> file "ietf-tpm-remote-attestation@2022-03-23.yang"
module ietf-tpm-remote-attestation {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-tpm-remote-attestation";
  prefix tpm;

  import ietf-yang-types {
    prefix yang;
  }
  import ietf-hardware {
    prefix hw;
  }
  import ietf-keystore {
    prefix ks;
  }
  import ietf-tcg-algs {
    prefix taa;
  }

  organization
    "IETF RATS (Remote ATtestation procedureS) Working Group";
  contact
    "WG Web   : <https://datatracker.ietf.org/wg/rats/>
    WG List  : <mailto:rats@ietf.org>
    Author   : Eric Voit <evoit@cisco.com>
    Author   : Henk Birkholz <henk.birkholz@sit.fraunhofer.de>
    Author   : Michael Eckel <michael.eckel@sit.fraunhofer.de>
    Author   : Shwetha Bhandari <shwetha.bhandari@thoughtspot.com>
    Author   : Bill Sulzen <bsulzen@cisco.com>
    Author   : Liang Xia (Frank) <frank.xialiang@huawei.com>
    Author   : Tom Laffey <tom.laffey@hpe.com>
    Author   : Guy Fedorkow <gfredorkow@juniper.net>";
  description
    "A YANG module to enable a TPM 1.2 and TPM 2.0 based
    remote attestation procedure using a challenge-response
    interaction model and the TPM 1.2 and TPM 2.0 Quote
    primitive operations.

    Copyright (c) 2022 IETF Trust and the persons identified
    as authors of the code. All rights reserved.
    Redistribution and use in source and binary forms, with or
    without modification, is permitted pursuant to, and subject to
    the license terms contained in, the Simplified BSD License set
    forth in Section 4.c of the IETF Trust's Legal Provisions
    Relating to IETF Documents
    (https://trustee.ietf.org/license-info).

    This version of this YANG module is part of RFC XXXX
```

(<https://www.rfc-editor.org/info/rfcXXXX>); see the RFC itself for full legal notices.

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 (RFC 2119) (RFC 8174) when, and only when, they appear in all capitals, as shown here.";

```
revision 2022-03-23 {
  description
    "Initial version";
  reference
    "RFC XXXX: A YANG Data Model for Challenge-Response-based Remote
    Attestation Procedures using TPMs";
}

/*****
/*   Features   */
*****/

feature mtpm {
  description
    "The device supports the remote attestation of multiple
    TPM based cryptoprocessors.";
}

feature bios {
  description
    "The device supports the bios logs.";
  reference
    "bios-log:
    https://trustedcomputinggroup.org/wp-content/uploads/
    PC-ClientSpecific_Platform_Profile_for_TPM_2p0_Systems_v51.pdf
    Section 9.4.5.2";
}

feature ima {
  description
    "The device supports Integrity Measurement Architecture logs.
    Many variants of IMA logs exist in the deployment. Each encodes
    the log entry contents as the specific measurements which get
    hashed into a PCRs as Evidence. See the reference below for
    one example of such an encoding.";
  reference
    "ima-log:
    https://www.trustedcomputinggroup.org/wp-content/uploads/
    TCG_IWG_CEL_v1_r0p41_pub.pdf Section 5.1.6";
}
```



```
    }

    feature netequip_boot {
      description
        "The device supports the netequip_boot logs.";
      reference
        "netequip-boot-log:
         RFC XXXX Appendix B";
    }

    /*****/
    /*  Typedefs  */
    /*****/

    typedef pcr {
      type uint8 {
        range "0..31";
      }
      description
        "Valid index number for a PCR. A {{TPM2.0}} compliant PCR index
         extends from 0-31. At this time a typical TPM would have no
         more than 32 PCRS.";
    }

    typedef compute-node-ref {
      type leafref {
        path "/tpm:rats-support-structures/tpm:compute-nodes"
          + "/tpm:compute-node/tpm:node-id";
      }
      description
        "This type is used to reference a hardware node. Note that an
         implementer might include an alternative leafref pointing to a
         different YANG module node specifying hardware structures.";
    }

    typedef certificate-name-ref {
      type leafref {
        path "/tpm:rats-support-structures/tpm:tpms/tpm:tpm"
          + "/tpm:certificates/tpm:certificate/tpm:name";
      }
      description
        "A type which allows identification of a TPM based certificate.";
    }

    /*****/
    /*  Identities  */
    /*****/
```

```
identity attested_event_log_type {
  description
    "Base identity allowing categorization of the reasons why an
    attested measurement has been taken on an Attester.";
}

identity ima {
  base attested_event_log_type;
  description
    "An event type recorded in IMA.";
}

identity bios {
  base attested_event_log_type;
  description
    "An event type associated with BIOS/UEFI.";
}

identity netequip_boot {
  base attested_event_log_type;
  description
    "An event type associated with Network Equipment Boot.";
}

/*****/
/*  Groupings  */
/*****/

grouping tpm20-hash-algo {
  description
    "The cryptographic algorithm used to hash the TPM2 PCRs. This
    must be from the list of platform supported options.";
  leaf tpm20-hash-algo {
    type identityref {
      base taa:hash;
    }
  }
  must '. = /tpm:rats-support-structures'
    + '/tpm:attester-supported-algos/tpm:tpm20-hash' {
    error-message "This platform does not support tpm20-hash-algo";
  }
  description
    "The hash scheme that is used to hash a TPM2.0 PCR. This
    must be one of those supported by a platform.
    Where this object does not appear, the default value of
    'taa:TPM_ALG_SHA256' will apply.";
}
}
```

```
grouping tpm12-hash-algo {
  description
    "The cryptographic algorithm used to hash the TPM1.2 PCRs.";
  leaf tpm12-hash-algo {
    type identityref {
      base taa:hash;
    }
    must '. = /tpm:rats-support-structures'
      + '/tpm:attester-supported-algos/tpm:tpm12-hash' {
      error-message "This platform does not support tpm12-hash-algo";
    }
    description
      "The hash scheme that is used to hash a TPM1.2 PCR. This
      MUST be one of those supported by a platform.
      Where this object does not appear, the default value of
      'taa:TPM_ALG_SHA1' will apply.";
  }
}

grouping nonce {
  description
    "A random number intended to guarantee freshness and for use
    as part of a replay-detection mechanism.";
  leaf nonce-value {
    type binary;
    mandatory true;
    description
      "A cryptographically generated random number which should
      not be predictable prior to its issuance from a random
      number generation function. The random number MUST be
      derived from an entropy source external to the Attester.

      Note that a nonce sent into a TPM will typically be 160 or 256
      binary digits long. (This is 20 or 32 bytes.) So if fewer
      binary digits are sent, this nonce object will be padded
      with leading zeros within Quotes returned from the TPM.
      Additionally if more bytes are sent, the nonce will be trimmed
      to the most significant binary digits.";
  }
}

grouping tpm12-pcr-selection {
  description
    "A Verifier can request one or more PCR values using its
    individually created Attestation Key Certificate (AC).
    The corresponding selection filter is represented in this
    grouping.";
  leaf-list pcr-index {
```

```
type pcr;
description
  "The numbers/indexes of the PCRs. In addition, any selection
  of PCRs MUST verify that the set of PCRs requested are a
  subset the set of PCRs exposed by in the leaf-list
  /tpm:rats-support-structures
  /tpm:tpms/tpm:tpm[name=current()]/tpm:tpm12-pcrs";
}
}

grouping tpm20-pcr-selection {
  description
    "A Verifier can acquire one or more PCR values, which are hashed
    together in a TPM2B_DIGEST coming from the TPM2. The selection
    list of desired PCRs and the Hash Algorithm is represented in
    this grouping.";
  list tpm20-pcr-selection {
    unique "tpm20-hash-algo";
    description
      "Specifies the list of PCRs and Hash Algorithms that can be
      returned within a TPM2B_DIGEST.";
    reference
      "TPM2.0-Structures:
      https://www.trustedcomputinggroup.org/wp-content/uploads/
      TPM-Rev-2.0-Part-2-Structures-01.38.pdf Section 10.9.7";
    uses tpm20-hash-algo;
    leaf-list pcr-index {
      type pcr;
      must '/tpm:rats-support-structures/tpm:tpms'
        + '/tpm:tpm[name = current()]'
        + '/tpm:tpm20-pcr-bank[pcr-index = current()]' {
        error-message "Acquiring this PCR index is not supported";
      }
      description
        "The numbers of the PCRs that which are being tracked
        with a hash based on the tpm20-hash-algo. In addition,
        any selection of PCRs MUST verify that the set of PCRs
        requested are a subset the set of PCR indexes exposed
        within /tpm:rats-support-structures/tpm:tpms
        /tpm:tpm[name=current()]/tpm:tpm20-pcr-bank
        /tpm:pcr-index";
    }
  }
}

grouping certificate-name-ref {
  description
    "Identifies a certificate in a keystore.";
```

```
    leaf certificate-name {
      type certificate-name-ref;
      mandatory true;
      description
        "Identifies a certificate in a keystore.";
    }
  }

  grouping tpm-name {
    description
      "A unique TPM on a device.";
    leaf name {
      type string;
      description
        "Unique system generated name for a TPM on a device.";
    }
  }

  grouping node-uptime {
    description
      "Uptime in seconds of the node.";
    leaf up-time {
      type uint32;
      description
        "Uptime in seconds of this node reporting its data";
    }
  }

  grouping tpml2-attestation {
    description
      "Contains an instance of TPM1.2 style signed cryptoprocessor
      measurements. It is supplemented by unsigned Attester
      information.";
    uses node-uptime;
    leaf TPM_QUOTE2 {
      type binary;
      description
        "Result of a TPM1.2 Quote2 operation. This includes PCRs,
        signatures, locality, the provided nonce and other data which
        can be further parsed to appraise the Attester.";
      reference
        "TPM1.2-Commands:
        TPM1.2 commands rev116 July 2007, Section 16.5
        https://trustedcomputinggroup.org/wp-content/uploads/TPM-Main-Part-3-Commands\_v1.2\_rev116\_01032011.pdf";
    }
  }
}
```

```
grouping tpm20-attestation {
  description
    "Contains an instance of TPM2 style signed cryptoprocessor
    measurements. It is supplemented by unsigned Attester
    information.";
  leaf TPMS_QUOTE_INFO {
    type binary;
    mandatory true;
    description
      "A hash of the latest PCR values (and the hash algorithm used)
      which have been returned from a Verifier for the selected PCRs
      and Hash Algorithms.";
    reference
      "TPM2.0-Structures:
      https://www.trustedcomputinggroup.org/wp-content/uploads/
      TPM-Rev-2.0-Part-2-Structures-01.38.pdf Section 10.12.1";
  }
  leaf quote-signature {
    type binary;
    description
      "Quote signature returned by TPM Quote. The signature was
      generated using the key associated with the
      certificate 'name'.";
    reference
      "TPM2.0-Structures:
      https://www.trustedcomputinggroup.org/wp-content/uploads/
      TPM-Rev-2.0-Part-2-Structures-01.38.pdf Section 11.2.1";
  }
  uses node-uptime;
  list unsigned-pcr-values {
    description
      "PCR values in each PCR bank. This might appear redundant with
      the TPM2B_DIGEST, but that digest is calculated across multiple
      PCRs. Having to verify across multiple PCRs does not
      necessarily make it easy for a Verifier to appraise just the
      minimum set of PCR information which has changed since the last
      received TPM2B_DIGEST. Put another way, why should a Verifier
      reconstruct the proper value of all PCR Quotes when only a
      single PCR has changed?
      To help this happen, if the Attester does know specific PCR
      values, the Attester can provide these individual values via
      'unsigned-pcr-values'. By comparing this information to
      what has previously been validated, it is possible for a
      Verifier to confirm the Attester's signature while eliminating
      significant processing. Note that there should never be a
      result where an unsigned PCR value differs from what may be
      reconstructed from the within the PCR quote and the event logs.
```

```
        If there is a difference, a signed result which has been
        verified from retrieved logs is considered definitive.";
uses tpm20-hash-algo;
list pcr-values {
  key "pcr-index";
  description
    "List of one PCR bank.";
  leaf pcr-index {
    type pcr;
    description
      "PCR index number.";
  }
  leaf pcr-value {
    type binary;
    description
      "PCR value.";
    reference
      "TPM2.0-Structures:
      https://www.trustedcomputinggroup.org/wp-content/uploads/
      TPM-Rev-2.0-Part-2-Structures-01.38.pdf Section 10.9.7";
  }
}
}
}

grouping log-identifier {
  description
    "Identifier for type of log to be retrieved.";
  leaf log-type {
    type identityref {
      base attested_event_log_type;
    }
    mandatory true;
    description
      "The corresponding measurement log type identity.";
  }
}

grouping boot-event-log {
  description
    "Defines a specific instance of an event log entry
    and corresponding to the information used to
    extend the PCR";
  leaf event-number {
    type uint32;
    description
      "Unique event number of this event which monotonically
      increases within a given event log. The maximum event
```

```
        number should not be reached, nor is wrapping back to
        an earlier number supported.";
    }
    leaf event-type {
        type uint32;
        description
            "BIOS Log Event Type:
            https://trustedcomputinggroup.org/wp-content/uploads/
            TCG_PCCClient_PFP_r1p05_v23_pub.pdf Section 10.4.1";
    }
    leaf pcr-index {
        type pcr;
        description
            "Defines the PCR index that this event extended";
    }
    list digest-list {
        description
            "Hash of event data";
        leaf hash-algo {
            type identityref {
                base taa:hash;
            }
            description
                "The hash scheme that is used to compress the event data in
                each of the leaf-list digest items.";
        }
        leaf-list digest {
            type binary;
            description
                "The hash of the event data using the algorithm of the
                'hash-algo' against 'event data'.";
        }
    }
    leaf event-size {
        type uint32;
        description
            "Size of the event data";
    }
    leaf-list event-data {
        type binary;
        description
            "The event data. This is a binary structure
            of size 'event-size'. For more on what
            might be recorded within this object
            see [bios-log] Section 9 which details
            viable events which might be recorded.";
    }
}
```



```
grouping bios-event-log {
  description
    "Measurement log created by the BIOS/UEFI.";
  list bios-event-entry {
    key "event-number";
    description
      "Ordered list of TCG described event log
       that extended the PCRs in the order they
       were logged";
    uses boot-event-log;
  }
}

grouping ima-event {
  description
    "Defines a hash log extend event for IMA measurements";
  reference
    "ima-log:
     https://www.trustedcomputinggroup.org/wp-content/uploads/TCG\_IWG\_CEL\_v1\_r0p41\_pub.pdf Section 4.3";
  leaf event-number {
    type uint64;
    description
      "Unique event number of this event which monotonically
       increases. The maximum event number should not be
       reached, nor is wrapping back to an earlier number
       supported.";
  }
  leaf ima-template {
    type string;
    description
      "Name of the template used for event logs
       for e.g. ima, ima-ng, ima-sig";
  }
  leaf filename-hint {
    type string;
    description
      "File name (including the path) that was measured.";
  }
  leaf filedata-hash {
    type binary;
    description
      "Hash of filedata as updated based upon the
       filedata-hash-algorithm";
  }
  leaf filedata-hash-algorithm {
    type string;
    description
```

```
        "Algorithm used for filedata-hash";
    }
    leaf template-hash-algorithm {
        type string;
        description
            "Algorithm used for template-hash";
    }
    leaf template-hash {
        type binary;
        description
            "hash(filedata-hash, filename-hint)";
    }
    leaf pcr-index {
        type pcr;
        description
            "Defines the PCR index that this event extended";
    }
    leaf signature {
        type binary;
        description
            "Digital file signature which provides a
            fingerprint for the file being measured.";
    }
}

grouping ima-event-log {
    description
        "Measurement log created by IMA.";
    list ima-event-entry {
        key "event-number";
        description
            "Ordered list of ima event logs by event-number";
        uses ima-event;
    }
}

grouping network-equipment-boot-event-log {
    description
        "Measurement log created by Network Equipment Boot. The Network
        Equipment Boot format is identical to the IMA format. In
        contrast to the IMA log, the Network Equipment Boot log
        includes every measurable event from an Attester, including
        the boot stages of BIOS, Bootloader, etc. In essence, the scope
        of events represented in this format combines the scope of BIOS
        events and IMA events.";
    list boot-event-entry {
        key "event-number";
        description
```

```

        "Ordered list of Network Equipment Boot event logs
        by event-number, using the IMA event format.";
    uses ima-event;
}
}

grouping event-logs {
    description
        "A selector for the log and its type.";
    choice attested_event_log_type {
        mandatory true;
        description
            "Event log type determines the event logs content.";
        case bios {
            if-feature "bios";
            description
                "BIOS/UEFI event logs";
            container bios-event-logs {
                description
                    "BIOS/UEFI event logs";
                uses bios-event-log;
            }
        }
        case ima {
            if-feature "ima";
            description
                "IMA event logs.";
            container ima-event-logs {
                description
                    "IMA event logs.";
                uses ima-event-log;
            }
        }
        case netequip_boot {
            if-feature "netequip_boot";
            description
                "Network Equipment Boot event logs";
            container boot-event-logs {
                description
                    "Network equipment boot event logs.";
                uses network-equipment-boot-event-log;
            }
        }
    }
}

/*****/
/*  RPC operations  */

```

```

/*****/

rpc tpm12-challenge-response-attestation {
  if-feature "taa:tpm12";
  description
    "This RPC accepts the input for TSS TPM 1.2 commands made to the
    attesting device.";
  input {
    container tpm12-attestation-challenge {
      description
        "This container includes every information element defined
        in the reference challenge-response interaction model for
        remote attestation. Corresponding values are based on
        TPM 1.2 structure definitions";
      uses tpm12-pcr-selection;
      uses nonce;
      leaf-list certificate-name {
        if-feature "tpm:mtpm";
        type certificate-name-ref;
        must "/tpm:rats-support-structures/tpm:tpms"
          + "/tpm:tpm[tpm:firmware-version='taa:tpm12']"
          + "/tpm:certificates/"
          + "/tpm:certificate[name=current()]" {
          error-message "Not an available TPM1.2 AIK certificate.";
        }
        description
          "When populated, the RPC will only get a Quote for the
          TPMs associated with these certificate(s).";
      }
    }
  }
  output {
    list tpm12-attestation-response {
      unique "certificate-name";
      description
        "The binary output of TPM 1.2 TPM_Quote/TPM_Quote2, including
        the PCR selection and other associated attestation evidence
        metadata";
      uses certificate-name-ref {
        description
          "Certificate associated with this tpm12-attestation.";
      }
      uses tpm12-attestation;
    }
  }
}

rpc tpm20-challenge-response-attestation {

```

```
if-feature "taa:tpm20";
description
  "This RPC accepts the input for TSS TPM 2.0 commands of the
  managed device. ComponentIndex from the hardware manager YANG
  module is used to refer to dedicated TPM in composite devices,
  e.g. smart NICs, is not covered.";
input {
  container tpm20-attestation-challenge {
    description
      "This container includes every information element defined
      in the reference challenge-response interaction model for
      remote attestation. Corresponding values are based on
      TPM 2.0 structure definitions";
    uses nonce;
    uses tpm20-pcr-selection;
    leaf-list certificate-name {
      if-feature "tpm:mtpm";
      type certificate-name-ref;
      must "/tpm:rats-support-structures/tpm:tpms"
        + "/tpm:tpm[tpm:firmware-version='taa:tpm20']"
        + "/tpm:certificates/"
        + "/tpm:certificate[name=current()]" {
        error-message "Not an available TPM2.0 AIK certificate.";
      }
      description
        "When populated, the RPC will only get a Quote for the
        TPMs associated with the certificates.";
    }
  }
}
output {
  list tpm20-attestation-response {
    unique "certificate-name";
    description
      "The binary output of TPM2b_Quote from one TPM of the
      node which identified by node-id. An TPMS_ATTEST structure
      including a length, encapsulated in a signature";
    uses certificate-name-ref {
      description
        "Certificate associated with this tpm20-attestation.";
    }
    uses tpm20-attestation;
  }
}

rpc log-retrieval {
  description
```

```
"Logs Entries are either identified via indices or via providing
the last line received. The number of lines returned can be
limited. The type of log is a choice that can be augmented.";
input {
  uses log-identifier;
  list log-selector {
    description
      "Only log entries which meet all the selection criteria
      provided are to be returned by the RPC output.";
    leaf-list name {
      type string;
      description
        "Name of one or more unique TPMs on a device. If this
        object exists, a selection should pull only the objects
        related to these TPM(s). If it does not exist, all
        qualifying TPMs that are 'hardware-based' equals true
        on the device are selected. When this selection
        criteria is provided, it will be considered as a logical
        AND with any other selection criteria provided.";
    }
    choice index-type {
      description
        "Last log entry received, log index number, or timestamp.";
      case last-entry {
        description
          "The last entry of the log already retrieved.";
        leaf last-entry-value {
          type binary;
          description
            "Content of a log event which matches 1:1 with a
            unique event record contained within the log. Log
            entries after this will be passed to the
            requester. Note: if log entry values are not unique,
            this MUST return an error.";
        }
      }
      case index {
        description
          "Numeric index of the last log entry retrieved, or
          zero.";
        leaf last-index-number {
          type uint64;
          description
            "The last numeric index number of a log entry.
            Zero means to start at the beginning of the log.
            Entries after this will be passed to the
            requester.";
        }
      }
    }
  }
}
```

```

    }
    case timestamp {
      leaf timestamp {
        type yang:date-and-time;
        description
          "Timestamp from which to start the extraction. The
           next log entry after this timestamp is to
           be sent.";
      }
      description
        "Timestamp from which to start the extraction.";
    }
  }
  leaf log-entry-quantity {
    type uint16;
    description
      "The number of log entries to be returned. If omitted, it
       means all of them.";
  }
}
}
output {
  container system-event-logs {
    description
      "The requested data of the measurement event logs";
    list node-data {
      unique "name";
      description
        "Event logs of a node in a distributed system
         identified by the node name";
      uses tpm-name;
      uses node-uptime;
      container log-result {
        description
          "The requested entries of the corresponding log.";
        uses event-logs;
      }
    }
  }
}
}

/*****/
/*  Config & Oper accessible nodes  */
/*****/

container rats-support-structures {
  description

```

```
    "The datastore definition enabling verifiers or relying
    parties to discover the information necessary to use the
    remote attestation RPCs appropriately.";
container compute-nodes {
  if-feature "tpm:mtpm";
  description
    "Holds the set of device subsystems/components in this
    composite device that support TPM operations.";
  list compute-node {
    key "node-id";
    unique "node-name";
    config false;
    min-elements 2;
    description
      "A component within this composite device which
      supports TPM operations.";
    leaf node-id {
      type string;
      description
        "ID of the compute node, such as Board Serial Number.";
    }
    leaf node-physical-index {
      if-feature "hw:entity-mib";
      type int32 {
        range "1..2147483647";
      }
      config false;
      description
        "The entPhysicalIndex for the compute node.";
      reference
        "RFC 6933: Entity MIB (Version 4) - entPhysicalIndex";
    }
    leaf node-name {
      type string;
      description
        "Name of the compute node.";
    }
    leaf node-location {
      type string;
      description
        "Location of the compute node, such as slot number.";
    }
  }
}
container tpms {
  description
    "Holds the set of TPMs within an Attester.";
  list tpm {
```



```
key "name";
unique "path";
description
  "A list of TPMs in this composite device that RATS
   can be conducted with.";
uses tpm-name;
leaf hardware-based {
  type boolean;
  config false;
  mandatory true;
  description
    "System generated indication of whether this is a
     hardware based TPM.";
}
leaf physical-index {
  if-feature "hw:entity-mib";
  type int32 {
    range "1..2147483647";
  }
  config false;
  description
    "The entPhysicalIndex for the TPM.";
  reference
    "RFC 6933: Entity MIB (Version 4) - entPhysicalIndex";
}
leaf path {
  type string;
  config false;
  description
    "Device path to a unique TPM on a device. This can change
     across reboots.";
}
leaf compute-node {
  if-feature "tpm:mtpm";
  type compute-node-ref;
  config false;
  mandatory true;
  description
    "Indicates the compute node measured by this TPM.";
}
leaf manufacturer {
  type string;
  config false;
  description
    "TPM manufacturer name.";
}
leaf firmware-version {
  type identityref {
```

```
    base taa:cryptoprocessor;
  }
  mandatory true;
  description
    "Identifies the cryptoprocessor API set supported. This
    is automatically configured by the device and should not
    be changed.";
}
uses tpm12-hash-algo {
  when "derived-from-or-self(firmware-version, 'taa:tpm12')";
  refine "tpm12-hash-algo" {
    description
      "The hash algorithm overwrites the default used for PCRs
      on this TPM1.2 compliant cryptoprocessor.";
  }
}
leaf-list tpm12-pcrs {
  when
    "derived-from-or-self(..../firmware-version, 'taa:tpm12')";
  type pcr;
  description
    "The PCRs which may be extracted from this TPM1.2
    compliant cryptoprocessor.";
}
list tpm20-pcr-bank {
  when
    "derived-from-or-self(..../firmware-version, 'taa:tpm20')";
  key "tpm20-hash-algo";
  description
    "Specifies the list of PCRs that may be extracted for
    a specific Hash Algorithm on this TPM2 compliant
    cryptoprocessor. A bank is a set of PCRs which are
    extended using a particular hash algorithm.";
  reference
    "TPM2.0-Structures:
    https://www.trustedcomputinggroup.org/wp-content/uploads/
    TPM-Rev-2.0-Part-2-Structures-01.38.pdf Section 10.9.7";
  leaf tpm20-hash-algo {
    type identityref {
      base taa:hash;
    }
    must '/tpm:rats-support-structures'
      + '/tpm:attester-supported-algos'
      + '/tpm:tpm20-hash' {
      error-message "This platform does not support tpm20-hash-algo";
    }
    description
      "The hash scheme actively being used to hash a
```

```
        one or more TPM2.0 PCRs.";
    }
    leaf-list pcr-index {
        type tpm:pcr;
        description
            "Defines what TPM2 PCRs are available to be extracted.";
    }
}
leaf status {
    type enumeration {
        enum operational {
            value 0;
            description
                "The TPM currently is running normally and
                 is ready to accept and process TPM quotes.";
            reference
                "TPM2.0-Arch:
                 https://trustedcomputinggroup.org/wp-content/uploads/TCG\_TPM2\_r1p59\_Part1\_Architecture\_pub.pdf
                 Section 12";
        }
        enum non-operational {
            value 1;
            description
                "TPM is in a state such as startup or shutdown which
                 precludes the processing of TPM quotes.";
        }
    }
}
config false;
mandatory true;
description
    "TPM chip self-test status.";
}
container certificates {
    description
        "The TPM's certificates, including EK certificates
         and Attestation Key certificates.";
    list certificate {
        key "name";
        description
            "Three types of certificates can be accessed via
             this statement, including Initial Attestation
             Key Certificate, Local Attestation Key Certificate or
             Endorsement Key Certificate.";
        leaf name {
            type string;
            description
                "An arbitrary name uniquely identifying a certificate
```

```
        associated within key within a TPM.";
    }
    leaf keystore-ref {
        if-feature "ks:asymmetric-keys";
        type leafref {
            path "/ks:keystore/ks:asymmetric-keys/ks:asymmetric-key"
                + "/ks:name";
        }
        description
            "A reference to a specific certificate of an
            asymmetric key in the Keystore.";
    }
    leaf type {
        type enumeration {
            enum endorsement-certificate {
                value 0;
                description
                    "Endorsement Key (EK) Certificate type.";
                reference
                    "TPM2.0-Key:
                    https://trustedcomputinggroup.org/wp-content/
                    uploads/TPM-2p0-Keys-for-Device-Identity-
                    and-Attestation_v1_r12_publ0082021.pdf
                    Section 3.11";
            }
            enum initial-attestation-certificate {
                value 1;
                description
                    "Initial Attestation key (IAK) Certificate type.";
                reference
                    "TPM2.0-Key:
                    https://trustedcomputinggroup.org/wp-content/
                    uploads/TPM-2p0-Keys-for-Device-Identity-
                    and-Attestation_v1_r12_publ0082021.pdf
                    Section 3.2";
            }
            enum local-attestation-certificate {
                value 2;
                description
                    "Local Attestation Key (LAK) Certificate type.";
                reference
                    "TPM2.0-Key:
                    https://trustedcomputinggroup.org/wp-content/
                    uploads/TPM-2p0-Keys-for-Device-Identity-
                    and-Attestation_v1_r12_publ0082021.pdf
                    Section 3.2";
            }
        }
    }
}
```

```
        description
          "Function supported by this certificate from within the
           TPM.";
      }
    }
  }
}
container attester-supported-algos {
  description
    "Identifies which TPM algorithms are available for use on an
     attesting platform.";
  leaf-list tpm12-asymmetric-signing {
    when "../..//tpm:tpms"
      + "/tpm:tpm[tpm:firmware-version='taa:tpm12']";
    type identityref {
      base taa:asymmetric;
    }
    description
      "Platform Supported TPM12 asymmetric algorithms.";
  }
  leaf-list tpm12-hash {
    when "../..//tpm:tpms"
      + "/tpm:tpm[tpm:firmware-version='taa:tpm12']";
    type identityref {
      base taa:hash;
    }
    description
      "Platform supported TPM12 hash algorithms.";
  }
  leaf-list tpm20-asymmetric-signing {
    when "../..//tpm:tpms"
      + "/tpm:tpm[tpm:firmware-version='taa:tpm20']";
    type identityref {
      base taa:asymmetric;
    }
    description
      "Platform Supported TPM20 asymmetric algorithms.";
  }
  leaf-list tpm20-hash {
    when "../..//tpm:tpms"
      + "/tpm:tpm[tpm:firmware-version='taa:tpm20']";
    type identityref {
      base taa:hash;
    }
    description
      "Platform supported TPM20 hash algorithms.";
  }
}
```

```

    }
  }
}
<CODE ENDS>

```

Figure 1

2.1.2. 'ietf-tcg-algs'

This document has encoded the TCG Algorithm definitions of [TCG-Algos], revision 1.32. By including this full table as a separate YANG file within this document, it is possible for other YANG models to leverage the contents of this model. Specific references to [RFC2104], [RFC8017], [ISO-IEC-9797-1], [ISO-IEC-9797-2], [ISO-IEC-10116], [ISO-IEC-10118-3], [ISO-IEC-14888-3], [ISO-IEC-15946-1], [ISO-IEC-18033-3], [IEEE-Std-1363-2000], [IEEE-Std-1363a-2004], [NIST-PUB-FIPS-202], [NIST-SP800-38C], [NIST-SP800-38D], [NIST-SP800-38F], [NIST-SP800-56A], [NIST-SP800-108], [bios-log], as well as Appendix A and Appendix B exist within the YANG Model.

2.1.2.1. Features

There are two types of features supported: 'TPM12' and 'TPM20'. Support for either of these features indicates that a cryptoprocessor supporting the corresponding type of TCG TPM API is present on an Attester. Most commonly, only one type of cryptoprocessor will be available on an Attester.

2.1.2.2. Identities

There are three types of identities in this model:

1. Cryptographic functions supported by a TPM algorithm; these include: 'asymmetric', 'symmetric', 'hash', 'signing', 'anonymous_signing', 'encryption_mode', 'method', and 'object_type'. The definitions of each of these are in Table 2 of [TCG-Algos].
2. API specifications for TPM types: 'tpm12' and 'tpm20'
3. Specific algorithm types: Each algorithm type defines what cryptographic functions may be supported, and on which type of API specification. It is not required that an implementation of a specific TPM will support all algorithm types. The contents of each specific algorithm mirrors what is in Table 3 of [TCG-Algos].

2.1.2.3. YANG Module

```

<CODE BEGINS> file "ietf-tcg-algs@2022-03-23.yang"
module ietf-tcg-algs {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-tcg-algs";
  prefix taa;

  organization
    "IETF RATS (Remote ATtestation procedureS) Working Group";
  contact
    "WG Web:  <https://datatracker.ietf.org/wg/rats/>
    WG List:  <mailto:rats@ietf.org>
    Author:   Eric Voit <mailto:evoit@cisco.com>";
  description
    "This module defines identities for asymmetric algorithms.

    Copyright (c) 2022 IETF Trust and the persons identified as
    authors of the code. All rights reserved.
    Redistribution and use in source and binary forms, with
    or without modification, is permitted pursuant to, and
    subject to the license terms contained in, the Revised
    BSD License set forth in Section 4.c of the IETF Trust's
    Legal Provisions Relating to IETF Documents
    (https://trustee.ietf.org/license-info).

    This version of this YANG module is part of RFC XXXX
    (https://www.rfc-editor.org/info/rfcXXXX); see the RFC itself
    for full legal notices.

    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 (RFC 2119)
    (RFC 8174) when, and only when, they appear in all
    capitals, as shown here.";

  revision 2022-03-23 {
    description
      "Initial version";
    reference
      "RFC XXXX: A YANG Data Model for Challenge-Response-based Remote
      Attestation Procedures using TPMs";
  }

  /*****/
  /*  Features  */
  /*****/

```

```
feature tpm12 {
  description
    "This feature indicates algorithm support for the TPM 1.2 API
    as per Section 4.8 of TPM1.2-Structures:
    TPM Main Part 2 TPM Structures
    https://trustedcomputinggroup.org/wp-content/uploads/TPM-
    Main-Part-2-TPM-Structures_v1.2_rev116_01032011.pdf";
}

feature tpm20 {
  description
    "This feature indicates algorithm support for the TPM 2.0 API
    as per Section 11.4 of Trusted Platform Module Library
    Part 1: Architecture. See TPM2.0-Arch:
    https://trustedcomputinggroup.org/wp-content/uploads/
    TCG_TPM2_rlp59_Part1_Architecture_pub.pdf";
}

/*****/
/* Identities */
/*****/

identity asymmetric {
  description
    "A TCG recognized asymmetric algorithm with a public and
    private key.";
  reference
    "TCG-Algos:TCG Algorithm Registry Rev1.32 Table 2,
    https://trustedcomputinggroup.org/resource/
    tcg-algorithm-registry/TCG-_Algorithm_Registry_rlp32_pub";
}

identity symmetric {
  description
    "A TCG recognized symmetric algorithm with only a private key.";
  reference
    "TCG-Algos:TCG Algorithm Registry Rev1.32 Table 2";
}

identity hash {
  description
    "A TCG recognized hash algorithm that compresses input data to
    a digest value or indicates a method that uses a hash.";
  reference
    "TCG-Algos:TCG Algorithm Registry Rev1.32 Table 2";
}

identity signing {
```



```
    description
      "A TCG recognized signing algorithm";
    reference
      "TCG-Algos:TCG Algorithm Registry Rev1.32 Table 2";
  }

  identity anonymous_signing {
    description
      "A TCG recognized anonymous signing algorithm.";
    reference
      "TCG-Algos:TCG Algorithm Registry Rev1.32 Table 2";
  }

  identity encryption_mode {
    description
      "A TCG recognized encryption mode.";
    reference
      "TCG-Algos:TCG Algorithm Registry Rev1.32 Table 2";
  }

  identity method {
    description
      "A TCG recognized method such as a mask generation function.";
    reference
      "TCG-Algos:TCG Algorithm Registry Rev1.32 Table 2";
  }

  identity object_type {
    description
      "A TCG recognized object type.";
    reference
      "TCG-Algos:TCG Algorithm Registry Rev1.32 Table 2";
  }

  identity cryptoprocessor {
    description
      "Base identity identifying a cryptoprocessor.";
  }

  identity tpml2 {
    if-feature "tpml2";
    base cryptoprocessor;
    description
      "Supportable by a TPM1.2.";
    reference
      "TPM1.2-Structures:
      https://trustedcomputinggroup.org/wp-content/uploads/
      TPM-Main-Part-2-TPM-Structures\_v1.2\_rev116\_01032011.pdf

```

```
        TPM_ALGORITHM_ID values, Section 4.8";
    }

    identity tpm20 {
        if-feature "tpm20";
        base cryptoprocessor;
        description
            "Supportable by a TPM2.";
        reference
            "TPM2.0-Structures:
            https://trustedcomputinggroup.org/wp-content/uploads/
            TPM-Rev-2.0-Part-2-Structures-01.38.pdf";
    }

    identity TPM_ALG_RSA {
        if-feature "tpm12 or tpm20";
        base tpm12;
        base tpm20;
        base asymmetric;
        base object_type;
        description
            "RSA algorithm";
        reference
            "TCG-Algos:TCG Algorithm Registry Rev1.32 Table 3 and
            RFC 8017. ALG_ID: 0x0001";
    }

    identity TPM_ALG_TDES {
        if-feature "tpm12";
        base tpm12;
        base symmetric;
        description
            "Block cipher with various key sizes (Triple Data Encryption
            Algorithm, commonly called Triple Data Encryption Standard)
            Note: was banned in TPM1.2 v94";
        reference
            "TCG-Algos:TCG Algorithm Registry Rev1.32 Table 3 and
            ISO/IEC 18033-3. ALG_ID: 0x0003";
    }

    identity TPM_ALG_SHA1 {
        if-feature "tpm12 or tpm20";
        base hash;
        base tpm12;
        base tpm20;
        description
            "SHA1 algorithm - Deprecated due to insufficient cryptographic
            protection. However, it is still useful for hash algorithms
```

```
        where protection is not required.";
    reference
        "TCG-Algos:TCG Algorithm Registry Rev1.32 Table 3 and
        ISO/IEC 10118-3. ALG_ID: 0x0004";
}

identity TPM_ALG_HMAC {
    if-feature "tpm12 or tpm20";
    base tpm12;
    base tpm20;
    base hash;
    base signing;
    description
        "Hash Message Authentication Code (HMAC) algorithm";
    reference
        "TCG-Algos:TCG Algorithm Registry Rev1.32 Table 3,
        ISO/IEC 9797-2 and RFC2104. ALG_ID: 0x0005";
}

identity TPM_ALG_AES {
    if-feature "tpm12";
    base tpm12;
    base symmetric;
    description
        "The AES algorithm with various key sizes";
    reference
        "TCG-Algos:TCG Algorithm Registry Rev1.32 Table 3,
        ISO/IEC 18033-3. ALG_ID: 0x0006";
}

identity TPM_ALG_MGF1 {
    if-feature "tpm20";
    base tpm20;
    base hash;
    base method;
    description
        "hash-based mask-generation function";
    reference
        "TCG-Algos:TCG Algorithm Registry Rev1.32 Table 3,
        IEEE Std 1363-2000 and IEEE Std 1363a-2004.
        ALG_ID: 0x0007";
}

identity TPM_ALG_KEYEDHASH {
    if-feature "tpm20";
    base tpm20;
    base hash;
    base object_type;
```

```
    description
      "An encryption or signing algorithm using a keyed hash.  These
      may use XOR for encryption or an HMAC for signing and may
      also refer to a data object that is neither signing nor
      encrypting.";
    reference
      "TCG-Algos:TCG Algorithm Registry Rev1.32 Table 3,
      ALG_ID: 0x0008";
  }

  identity TPM_ALG_XOR {
    if-feature "tpm12 or tpm20";
    base tpm12;
    base tpm20;
    base hash;
    base symmetric;
    description
      "The XOR encryption algorithm.";
    reference
      "TCG-Algos:TCG Algorithm Registry Rev1.32 Table 3.
      ALG_ID: 0x000A";
  }

  identity TPM_ALG_SHA256 {
    if-feature "tpm20";
    base tpm20;
    base hash;
    description
      "The SHA 256 algorithm";
    reference
      "TCG-Algos:TCG Algorithm Registry Rev1.32 Table 3 and
      ISO/IEC 10118-3. ALG_ID: 0x000B";
  }

  identity TPM_ALG_SHA384 {
    if-feature "tpm20";
    base tpm20;
    base hash;
    description
      "The SHA 384 algorithm";
    reference
      "TCG-Algos:TCG Algorithm Registry Rev1.32 Table 3 and
      ISO/IEC 10118-3. ALG_ID: 0x000C";
  }

  identity TPM_ALG_SHA512 {
    if-feature "tpm20";
    base tpm20;
```

```
    base hash;
    description
        "The SHA 512 algorithm";
    reference
        "TCG-Algos:TCG Algorithm Registry Rev1.32 Table 3 and
        ISO/IEC 10118-3. ALG_ID: 0x000D";
}

identity TPM_ALG_NULL {
    if-feature "tpm20";
    base tpm20;
    description
        "NULL algorithm";
    reference
        "TCG-Algos:TCG Algorithm Registry Rev1.32 Table 3.
        ALG_ID: 0x0010";
}

identity TPM_ALG_SM3_256 {
    if-feature "tpm20";
    base tpm20;
    base hash;
    description
        "The SM3 hash algorithm.";
    reference
        "TCG-Algos:TCG Algorithm Registry Rev1.32 Table 3 and
        ISO/IEC 10118-3:2018. ALG_ID: 0x0012";
}

identity TPM_ALG_SM4 {
    if-feature "tpm20";
    base tpm20;
    base symmetric;
    description
        "SM4 symmetric block cipher";
    reference
        "TCG-Algos:TCG Algorithm Registry Rev1.32 Table 3.
        ALG_ID: 0x0013";
}

identity TPM_ALG_RSASSA {
    if-feature "tpm20";
    base tpm20;
    base asymmetric;
    base signing;
    description
        "RFC 8017 Signature algorithm defined in section 8.2
        (RSASSAPKCS1-v1_5)";
```

```
    reference
      "TCG-Algos:TCG Algorithm Registry Rev1.32 Table 3 and
      RFC 8017. ALG_ID: 0x0014";
  }

  identity TPM_ALG_RSAES {
    if-feature "tpm20";
    base tpm20;
    base asymmetric;
    base encryption_mode;
    description
      "RFC 8017 Signature algorithm defined in section 7.2
      (RSAES-PKCS1-v1_5)";
    reference
      "TCG-Algos:TCG Algorithm Registry Rev1.32 Table 3 and
      RFC 8017. ALG_ID: 0x0015";
  }

  identity TPM_ALG_RSAPSS {
    if-feature "tpm20";
    base tpm20;
    base asymmetric;
    base signing;
    description
      "Padding algorithm defined in section 8.1 (RSASSA PSS)";
    reference
      "TCG-Algos:TCG Algorithm Registry Rev1.32 Table 3 and
      RFC 8017. ALG_ID: 0x0016";
  }

  identity TPM_ALG_OAEP {
    if-feature "tpm20";
    base tpm20;
    base asymmetric;
    base encryption_mode;
    description
      "Padding algorithm defined in section 7.1 (RSASSA OAEP)";
    reference
      "TCG-Algos:TCG Algorithm Registry Rev1.32 Table 3 and
      RFC 8017. ALG_ID: 0x0017";
  }

  identity TPM_ALG_ECDSA {
    if-feature "tpm20";
    base tpm20;
    base asymmetric;
    base signing;
    description
```

```
        "Signature algorithm using elliptic curve cryptography (ECC)";
    reference
        "TCG-Algos:TCG Algorithm Registry Rev1.32 Table 3 and
        ISO/IEC 14888-3. ALG_ID: 0x0018";
}

identity TPM_ALG_ECDH {
    if-feature "tpm20";
    base tpm20;
    base asymmetric;
    base method;
    description
        "Secret sharing using ECC";
    reference
        "TCG-Algos:TCG Algorithm Registry Rev1.32 Table 3 and
        NIST SP800-56A. ALG_ID: 0x0019";
}

identity TPM_ALG_ECDSA {
    if-feature "tpm20";
    base tpm20;
    base asymmetric;
    base signing;
    base anonymous_signing;
    description
        "Elliptic-curve based anonymous signing scheme";
    reference
        "TCG-Algos:TCG Algorithm Registry Rev1.32 Table 3 and
        TCG TPM 2.0 library specification. ALG_ID: 0x001A";
}

identity TPM_ALG_SM2 {
    if-feature "tpm20";
    base tpm20;
    base asymmetric;
    base signing;
    base encryption_mode;
    base method;
    description
        "SM2 - depending on context, either an elliptic-curve based,
        signature algorithm, an encryption scheme, or a key exchange
        protocol";
    reference
        "TCG-Algos:TCG Algorithm Registry Rev1.32 Table 3.
        ALG_ID: 0x001B";
}

identity TPM_ALG_ECSCHNORR {
```

```
    if-feature "tpm20";
    base tpm20;
    base asymmetric;
    base signing;
    description
        "Elliptic-curve based Schnorr signature";
    reference
        "TCG-Algos:TCG Algorithm Registry Rev1.32 Table 3.
        ALG_ID: 0x001C";
}

identity TPM_ALG_ECMQV {
    if-feature "tpm20";
    base tpm20;
    base asymmetric;
    base method;
    description
        "Two-phase elliptic-curve key";
    reference
        "TCG-Algos:TCG Algorithm Registry Rev1.32 Table 3 and
        NIST SP800-56A. ALG_ID: 0x001D";
}

identity TPM_ALG_KDF1_SP800_56A {
    if-feature "tpm20";
    base tpm20;
    base hash;
    base method;
    description
        "Concatenation key derivation function";
    reference
        "TCG-Algos:TCG Algorithm Registry Rev1.32 Table 3 and
        NIST SP800-56A (approved alternative1) section 5.8.1.
        ALG_ID: 0x0020";
}

identity TPM_ALG_KDF2 {
    if-feature "tpm20";
    base tpm20;
    base hash;
    base method;
    description
        "Key derivation function";
    reference
        "TCG-Algos:TCG Algorithm Registry Rev1.32 Table 3 and
        IEEE 1363a-2004 KDF2 section 13.2. ALG_ID: 0x0021";
}
```



```
identity TPM_ALG_KDF1_SP800_108 {
  base TPM_ALG_KDF2;
  description
    "A key derivation method";
  reference
    "TCG-Algos:TCG Algorithm Registry Rev1.32 Table 3 and
    NIST SP800-108 - Section 5.1 KDF. ALG_ID: 0x0022";
}

identity TPM_ALG_ECC {
  if-feature "tpm20";
  base tpm20;
  base asymmetric;
  base object_type;
  description
    "Prime field ECC";
  reference
    "TCG-Algos:TCG Algorithm Registry Rev1.32 Table 3 and
    ISO/IEC 15946-1. ALG_ID: 0x0023";
}

identity TPM_ALG_SYMCIPHER {
  if-feature "tpm20";
  base tpm20;
  base symmetric;
  base object_type;
  description
    "Object type for a symmetric block cipher";
  reference
    "TCG-Algos:TCG Algorithm Registry Rev1.32 Table 3 and
    TCG TPM 2.0 library specification. ALG_ID: 0x0025";
}

identity TPM_ALG_CAMELLIA {
  if-feature "tpm20";
  base tpm20;
  base symmetric;
  description
    "The Camellia algorithm";
  reference
    "TCG-Algos:TCG Algorithm Registry Rev1.32 Table 3 and
    ISO/IEC 18033-3. ALG_ID: 0x0026";
}

identity TPM_ALG_SHA3_256 {
  if-feature "tpm20";
  base tpm20;
  base hash;
```

```
    description
      "ISO/IEC 10118-3 - the SHA 256 algorithm";
    reference
      "TCG-Algos:TCG Algorithm Registry Rev1.32 Table 3 and
      NIST PUB FIPS 202. ALG_ID: 0x0027";
  }

  identity TPM_ALG_SHA3_384 {
    if-feature "tpm20";
    base tpm20;
    base hash;
    description
      "The SHA 384 algorithm";
    reference
      "TCG-Algos:TCG Algorithm Registry Rev1.32 Table 3 and
      NIST PUB FIPS 202. ALG_ID: 0x0028";
  }

  identity TPM_ALG_SHA3_512 {
    if-feature "tpm20";
    base tpm20;
    base hash;
    description
      "The SHA 512 algorithm";
    reference
      "TCG-Algos:TCG Algorithm Registry Rev1.32 Table 3 and
      NIST PUB FIPS 202. ALG_ID: 0x0029";
  }

  identity TPM_ALG_CMAC {
    if-feature "tpm20";
    base tpm20;
    base symmetric;
    base signing;
    description
      "block Cipher-based Message Authentication Code (CMAC)";
    reference
      "TCG-Algos:TCG Algorithm Registry Rev1.32 Table 3 and
      ISO/IEC 9797-1:2011 Algorithm 5. ALG_ID: 0x003F";
  }

  identity TPM_ALG_CTR {
    if-feature "tpm20";
    base tpm20;
    base symmetric;
    base encryption_mode;
    description
      "Counter mode";
```

```
    reference
      "TCG-Algos:TCG Algorithm Registry Rev1.32  Table 3 and
      ISO/IEC 10116. ALG_ID: 0x0040";
  }

  identity TPM_ALG_OFB {
    base tpm20;
    base symmetric;
    base encryption_mode;
    description
      "Output Feedback mode";
    reference
      "TCG-Algos:TCG Algorithm Registry Rev1.32  Table 3 and
      ISO/IEC 10116. ALG_ID: 0x0041";
  }

  identity TPM_ALG_CBC {
    if-feature "tpm20";
    base tpm20;
    base symmetric;
    base encryption_mode;
    description
      "Cipher Block Chaining mode";
    reference
      "TCG-Algos:TCG Algorithm Registry Rev1.32  Table 3 and
      ISO/IEC 10116. ALG_ID: 0x0042";
  }

  identity TPM_ALG_CFB {
    if-feature "tpm20";
    base tpm20;
    base symmetric;
    base encryption_mode;
    description
      "Cipher Feedback mode";
    reference
      "TCG-Algos:TCG Algorithm Registry Rev1.32  Table 3 and
      ISO/IEC 10116. ALG_ID: 0x0043";
  }

  identity TPM_ALG_ECB {
    if-feature "tpm20";
    base tpm20;
    base symmetric;
    base encryption_mode;
    description
      "Electronic Codebook mode";
    reference
```

```
        "TCG-Algos:TCG Algorithm Registry Rev1.32  Table 3 and
        ISO/IEC 10116. ALG_ID: 0x0044";
    }

    identity TPM_ALG_CCM {
        if-feature "tpm20";
        base tpm20;
        base symmetric;
        base signing;
        base encryption_mode;
        description
            "Counter with Cipher Block Chaining-Message Authentication
            Code (CCM)";
        reference
            "TCG-Algos:TCG Algorithm Registry Rev1.32  Table 3 and
            NIST SP800-38C. ALG_ID: 0x0050";
    }

    identity TPM_ALG_GCM {
        if-feature "tpm20";
        base tpm20;
        base symmetric;
        base signing;
        base encryption_mode;
        description
            "Galois/Counter Mode (GCM)";
        reference
            "TCG-Algos:TCG Algorithm Registry Rev1.32  Table 3 and
            NIST SP800-38D. ALG_ID: 0x0051";
    }

    identity TPM_ALG_KW {
        if-feature "tpm20";
        base tpm20;
        base symmetric;
        base signing;
        base encryption_mode;
        description
            "AES Key Wrap (KW)";
        reference
            "TCG-Algos:TCG Algorithm Registry Rev1.32  Table 3 and
            NIST SP800-38F. ALG_ID: 0x0052";
    }

    identity TPM_ALG_KWP {
        if-feature "tpm20";
        base tpm20;
        base symmetric;
```

```
    base signing;
    base encryption_mode;
    description
        "AES Key Wrap with Padding (KWP)";
    reference
        "TCG-Algos:TCG Algorithm Registry Rev1.32 Table 3 and
        NIST SP800-38F. ALG_ID: 0x0053";
}

identity TPM_ALG_EAX {
    if-feature "tpm20";
    base tpm20;
    base symmetric;
    base signing;
    base encryption_mode;
    description
        "Authenticated-Encryption Mode";
    reference
        "TCG-Algos:TCG Algorithm Registry Rev1.32 Table 3 and
        NIST SP800-38F. ALG_ID: 0x0054";
}

identity TPM_ALG_EDDSA {
    if-feature "tpm20";
    base tpm20;
    base asymmetric;
    base signing;
    description
        "Edwards-curve Digital Signature Algorithm (PureEdDSA)";
    reference
        "TCG-Algos:TCG Algorithm Registry Rev1.32 Table 3 and
        RFC 8032. ALG_ID: 0x0060";
}
}
<CODE ENDS>
```

Note that not all cryptographic functions are required for use by ietf-tpm-remote-attestation.yang. However the full definition of Table 3 of [TCG-Algos] will allow use by additional YANG specifications.

3. IANA Considerations

This document registers the following namespace URIs in the [xml-registry] as per [RFC3688]:

URI: urn:ietf:params:xml:ns:yang:ietf-tpm-remote-attestation

Registrant Contact: The IESG.

XML: N/A; the requested URI is an XML namespace.

URI: urn:ietf:params:xml:ns:yang:ietf-tcg-algs

Registrant Contact: The IESG.

XML: N/A; the requested URI is an XML namespace.

This document registers the following YANG modules in the registry [yang-parameters] as per Section 14 of [RFC6020]:

Name: ietf-tpm-remote-attestation

Namespace: urn:ietf:params:xml:ns:yang:ietf-tpm-remote-attestation

Prefix: tpm

Reference: draft-ietf-rats-yang-tpm-charra (RFC form)

Name: ietf-tcg-algs

Namespace: urn:ietf:params:xml:ns:yang:ietf-tcg-algs

Prefix: taa

Reference: draft-ietf-rats-yang-tpm-charra (RFC form)

4. Security Considerations

The YANG module ietf-tpm-remote-attestation.yang specified in this document defines a schema for data that is designed to be accessed via network management protocols such as NETCONF [RFC6241] or RESTCONF [RFC8040]. The lowest NETCONF layer is the secure transport layer, and the mandatory-to-implement secure transport is Secure Shell (SSH) [RFC6242]. The lowest RESTCONF layer is HTTPS, and the mandatory-to-implement secure transport is TLS [RFC8446].

There are a number of data nodes defined in this YANG module that are writable/creatable/deletable (i.e., `_config true`, which is the default). These data nodes may be considered sensitive or vulnerable in some network environments. Write operations (e.g., `_edit-config`) to these data nodes without proper protection can have a negative effect on network operations. These are the subtrees and data nodes as well as their sensitivity/vulnerability:

Container `'/rats-support-structures/attester-supported-algos'`: `'tpm12-asymmetric-signing'`, `'tpm12-hash'`, `'tpm20-asymmetric-signing'`, and `'tpm20-hash'`. All could be populated with algorithms that are not supported by the underlying physical TPM installed by the equipment vendor. A vendor should restrict the ability to configure unsupported algorithms.

Container: `'/rats-support-structures/tpms'`: `'name'`: Although shown as `'rw'`, it is system generated. Therefore, it should not be possible for an operator to add or remove a TPM from the configuration.

`'tpm20-pcr-bank'`: It is possible to configure PCRs for extraction which are not being extended by system software. This could unnecessarily use TPM resources.

`'certificates'`: It is possible to provision a certificate which does not correspond to an Attestation Identity Key (AIK) within the TPM 1.2, or an Attestation Key (AK) within the TPM 2.0 respectively. In such a case, calls to an RPC requesting this specific certificate could result in either no response or a response for an unexpected TPM.

RPC `'tpm12-challenge-response-attestation'`: The receiver of the RPC response must verify that the certificate is for an active AIK, i.e., the certificate has been confirmed by a third party as being able to support Attestation on the targeted TPM 1.2.

RPC `'tpm20-challenge-response-attestation'`: The receiver of the RPC response must verify that the certificate is for an active AK, i.e., the private key confirmation of the quote signature within the RPC response has been confirmed by a third party to belong to an entity legitimately able to perform Attestation on the targeted TPM 2.0.

RPC `'log-retrieval'`: Requesting a large volume of logs from the attester could require significant system resources and create a denial of service.

Information collected through the RPCs above could reveal that specific versions of software and configurations of endpoints that could identify vulnerabilities on those systems. Therefore, RPCs should be protected by NACM [RFC8341] with a default setting of deny-all to limit the extraction of attestation data by only authorized Verifiers.

For the YANG module `ietf-tcg-algs.yang`, please use care when selecting specific algorithms. The introductory section of [TCG-Algos] highlights that some algorithms should be considered legacy, and recommends implementers and adopters diligently evaluate available information such as governmental, industrial, and academic research before selecting an algorithm for use.

5. References

5.1. Normative References

- [bios-log] "TCG PC Client Platform Firmware Profile Specification, Section 9.4.5.2", n.d., <https://trustedcomputinggroup.org/wp-content/uploads/PC-ClientSpecific_Platform_Profile_for_TPM_2p0_Systems_v51.pdf>.
- [BIOS-Log-Event-Type] "TCG PC Client Platform Firmware Profile Specification", n.d., <https://trustedcomputinggroup.org/wp-content/uploads/TCG_PCClient_PFP_r1p05_v23_pub.pdf>.
- [cel] "Canonical Event Log Format, Section 4.3", n.d., <https://trustedcomputinggroup.org/wp-content/uploads/TCG_IWG_CEL_v1_r0p41_pub.pdf>.
- [I-D.ietf-netconf-keystore] Watsen, K., "A YANG Data Model for a Keystore", Work in Progress, Internet-Draft, draft-ietf-netconf-keystore-24, 7 March 2022, <<https://www.ietf.org/archive/id/draft-ietf-netconf-keystore-24.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-15, 8 February 2022, <<https://www.ietf.org/archive/id/draft-ietf-rats-architecture-15.txt>>.
- [I-D.ietf-rats-tpm-based-network-device-attest] Fedorkow, G., Voit, E., and J. Fitzgerald-McKay, "TPM-based Network Device Remote Integrity Verification", Work in Progress, Internet-Draft, draft-ietf-rats-tpm-based-network-device-attest-14, 22 March 2022, <<https://www.ietf.org/archive/id/draft-ietf-rats-tpm-based-network-device-attest-14.txt>>.

- [IEEE-Std-1363-2000]
"IEEE 1363-2000 - IEEE Standard Specifications for Public-Key Cryptography", n.d.,
<<https://standards.ieee.org/standard/1363-2000.html>>.
- [IEEE-Std-1363a-2004]
"1363a-2004 - IEEE Standard Specifications for Public-Key Cryptography - Amendment 1: Additional Techniques", n.d.,
<<https://ieeexplore.ieee.org/document/1335427>>.
- [ISO-IEC-10116]
"ISO/IEC 10116:2017 - Information technology", n.d.,
<<https://www.iso.org/standard/64575.html>>.
- [ISO-IEC-10118-3]
"Dedicated hash-functions - ISO/IEC 10118-3:2018", n.d.,
<<https://www.iso.org/standard/67116.html>>.
- [ISO-IEC-14888-3]
"ISO/IEC 14888-3:2018 - Digital signatures with appendix", n.d., <<https://www.iso.org/standard/76382.html>>.
- [ISO-IEC-15946-1]
"ISO/IEC 15946-1:2016 - Information technology", n.d.,
<<https://www.iso.org/standard/65480.html>>.
- [ISO-IEC-18033-3]
"ISO/IEC 18033-3:2010 - Encryption algorithms", n.d.,
<<https://www.iso.org/standard/54531.html>>.
- [ISO-IEC-9797-1]
"Message Authentication Codes (MACs) - ISO/IEC 9797-1:2011", n.d.,
<<https://www.iso.org/standard/50375.html>>.
- [ISO-IEC-9797-2]
"Message Authentication Codes (MACs) - ISO/IEC 9797-2:2011", n.d.,
<<https://www.iso.org/standard/51618.html>>.
- [NIST-PUB-FIPS-202]
"SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions", n.d.,
<<https://csrc.nist.gov/publications/detail/fips/202/final>>.

- [NIST-SP800-108]
"Recommendation for Key Derivation Using Pseudorandom Functions", n.d.,
<<https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-108.pdf>>.
- [NIST-SP800-38C]
"Recommendation for Block Cipher Modes of Operation: the CCM Mode for Authentication and Confidentiality", n.d.,
<<https://csrc.nist.gov/publications/detail/sp/800-38c/final>>.
- [NIST-SP800-38D]
"Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC", n.d.,
<<https://csrc.nist.gov/publications/detail/sp/800-38d/final>>.
- [NIST-SP800-38F]
"Recommendation for Block Cipher Modes of Operation: Methods for Key Wrapping", n.d.,
<<https://csrc.nist.gov/publications/detail/sp/800-38f/final>>.
- [NIST-SP800-56A]
"Recommendation for Pair-Wise Key-Establishment Schemes Using Discrete Logarithm Cryptography", n.d.,
<<https://csrc.nist.gov/publications/detail/sp/800-56a/rev-3/final>>.
- [RFC2104] Krawczyk, H., Bellare, M., and R. Canetti, "HMAC: Keyed-Hashing for Message Authentication", RFC 2104, DOI 10.17487/RFC2104, February 1997,
<<https://www.rfc-editor.org/info/rfc2104>>.
- [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>>.
- [RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, DOI 10.17487/RFC3688, January 2004,
<<https://www.rfc-editor.org/info/rfc3688>>.
- [RFC6020] Bjorklund, M., Ed., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", RFC 6020, DOI 10.17487/RFC6020, October 2010,
<<https://www.rfc-editor.org/info/rfc6020>>.

- [RFC6241] Enns, R., Ed., Bjorklund, M., Ed., Schoenwaelder, J., Ed., and A. Bierman, Ed., "Network Configuration Protocol (NETCONF)", RFC 6241, DOI 10.17487/RFC6241, June 2011, <<https://www.rfc-editor.org/info/rfc6241>>.
- [RFC6242] Wasserman, M., "Using the NETCONF Protocol over Secure Shell (SSH)", RFC 6242, DOI 10.17487/RFC6242, June 2011, <<https://www.rfc-editor.org/info/rfc6242>>.
- [RFC6933] Bierman, A., Romascanu, D., Quittek, J., and M. Chandramouli, "Entity MIB (Version 4)", RFC 6933, DOI 10.17487/RFC6933, May 2013, <<https://www.rfc-editor.org/info/rfc6933>>.
- [RFC6991] Schoenwaelder, J., Ed., "Common YANG Data Types", RFC 6991, DOI 10.17487/RFC6991, July 2013, <<https://www.rfc-editor.org/info/rfc6991>>.
- [RFC8017] Moriarty, K., Ed., Kaliski, B., Jonsson, J., and A. Rusch, "PKCS #1: RSA Cryptography Specifications Version 2.2", RFC 8017, DOI 10.17487/RFC8017, November 2016, <<https://www.rfc-editor.org/info/rfc8017>>.
- [RFC8032] Josefsson, S. and I. Liusvaara, "Edwards-Curve Digital Signature Algorithm (EdDSA)", RFC 8032, DOI 10.17487/RFC8032, January 2017, <<https://www.rfc-editor.org/info/rfc8032>>.
- [RFC8040] Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", RFC 8040, DOI 10.17487/RFC8040, January 2017, <<https://www.rfc-editor.org/info/rfc8040>>.
- [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/info/rfc8174>>.
- [RFC8341] Bierman, A. and M. Bjorklund, "Network Configuration Access Control Model", STD 91, RFC 8341, DOI 10.17487/RFC8341, March 2018, <<https://www.rfc-editor.org/info/rfc8341>>.
- [RFC8348] Bierman, A., Bjorklund, M., Dong, J., and D. Romascanu, "A YANG Data Model for Hardware Management", RFC 8348, DOI 10.17487/RFC8348, March 2018, <<https://www.rfc-editor.org/info/rfc8348>>.

- [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>>.
- [TCG-Algos]
"TCG Algorithm Registry", n.d.,
<https://trustedcomputinggroup.org/wp-content/uploads/TCG-Algorithm_Registry_r1p32_pub.pdf>.
- [TPM1.2] TCG, ., "TPM 1.2 Main Specification", 2 October 2003,
<<https://trustedcomputinggroup.org/resource/tpm-main-specification/>>.
- [TPM1.2-Commands]
"TPM Main Part 3 Commands", n.d.,
<https://trustedcomputinggroup.org/wp-content/uploads/TPM-Main-Part-3-Commands_v1.2_rev116_01032011.pdf>.
- [TPM1.2-Structures]
"TPM Main Part 2 TPM Structures", n.d.,
<https://trustedcomputinggroup.org/wp-content/uploads/TPM-Main-Part-2-TPM-Structures_v1.2_rev116_01032011.pdf>.
- [TPM2.0] TCG, ., "TPM 2.0 Library Specification", 15 March 2013,
<<https://trustedcomputinggroup.org/resource/tpm-library-specification/>>.
- [TPM2.0-Arch]
"Trusted Platform Module Library - Part 1: Architecture", n.d., <https://trustedcomputinggroup.org/wp-content/uploads/TCG_TPM2_r1p59_Part1_Architecture_pub.pdf>.
- [TPM2.0-Key]
TCG, ., "TPM 2.0 Keys for Device Identity and Attestation, Rev12", 8 October 2021,
<https://trustedcomputinggroup.org/wp-content/uploads/TPM-2p0-Keys-for-Device-Identity-and-Attestation_v1_r12_pub10082021.pdf>.
- [TPM2.0-Structures]
"Trusted Platform Module Library - Part 2: Structures", n.d., <<https://trustedcomputinggroup.org/wp-content/uploads/TPM-Rev-2.0-Part-2-Structures-01.38.pdf>>.
- [UEFI-Secure-Boot]
"Unified Extensible Firmware Interface (UEFI) Specification Version 2.9 (March 2021), Section 32.1

(Secure Boot)", n.d.,
<https://uefi.org/sites/default/files/resources/UEFI_Spec_2_9_2021_03_18.pdf>.

5.2. Informative References

- [I-D.ietf-rats-reference-interaction-models]
Birkholz, H., Eckel, M., Pan, W., and E. Voit, "Reference Interaction Models for Remote Attestation Procedures", Work in Progress, Internet-Draft, draft-ietf-rats-reference-interaction-models-05, 26 January 2022, <<https://www.ietf.org/archive/id/draft-ietf-rats-reference-interaction-models-05.txt>>.
- [IMA-Kernel-Source]
"Linux Integrity Measurement Architecture (IMA): Kernel Sourcecode", n.d., <<https://github.com/torvalds/linux/blob/df0cc57e057f18e44dac8e6c18aba47ab53202f9/security/integrity/ima/>>.
- [NIST-915121]
"True Randomness Can't be Left to Chance: Why entropy is important for information security", n.d., <https://tsapps.nist.gov/publication/get_pdf.cfm?pub_id=915121>.
- [xml-registry]
"IETF XML Registry", n.d., <<https://www.iana.org/assignments/xml-registry/xml-registry.xhtml>>.
- [yang-parameters]
"YANG Parameters", n.d., <<https://www.iana.org/assignments/yang-parameters/yang-parameters.xhtml>>.

Appendix A. Integrity Measurement Architecture (IMA)

IMA extends the principles of Measured Boot [TPM2.0-Arch] and Secure Boot [UEFI-Secure-Boot] to the Linux operating system, applying it to operating system applications and files. IMA has been part of the Linux integrity subsystem of the Linux kernel since 2009 (kernel version 2.6.30). The IMA mechanism represented by the YANG module in this specification is rooted in the kernel version 5.16 [IMA-Kernel-Source]. IMA enables the protection of system integrity by collecting (commonly referred to as measuring) and storing measurements (called Claims in the context of IETF RATS) of files before execution so that these measurements can be used later, at

system runtime, in remote attestation procedures. IMA acts in support of the appraisal of Evidence (which includes measurement Claims) by leveraging reference integrity measurements stored in extended file attributes.

In support of the appraisal of Evidence, IMA maintains an ordered list of measurements in kernel-space, the Stored Measurement Log (SML), for all files that have been measured before execution since the operating system was started. Although IMA can be used without a TPM, it is typically used in conjunction with a TPM to anchor the integrity of the SML in a hardware-protected secure storage location, i.e., Platform Configuration Registers (PCRs) provided by TPMs. IMA provides the SML in both binary and ASCII representations in the Linux security file system `_securityfs_ (/sys/kernel/security/ima/)`.

IMA templates define the format of the SML, i.e., which fields are included in a log record. Examples are file path, file hash, user ID, group ID, file signature, and extended file attributes. IMA comes with a set of predefined template formats and also allows a custom format, i.e., a format consisting of template fields supported by IMA. Template usage is typically determined by boot arguments passed to the kernel. Alternatively, the format can also be hard-coded into custom kernels. IMA templates and fields are extensible in the kernel source code. As a result, more template fields can be added in the future.

IMA policies define which files are measured using the IMA policy language. Built-in policies can be passed as boot arguments to the kernel. Custom IMA policies can be defined once during runtime or be hard-coded into a custom kernel. If no policy is defined, no measurements are taken and IMA is effectively disabled.

A comprehensive description of the content fields in native Linux IMA TLV format can be found in Table 16 of the Canonical Event Log (CEL) specification [cel]. The CEL specification also illustrates the use of templates to enable extended or customized IMA TLV formats in Section 5.1.6.

Appendix B. IMA for Network Equipment Boot Logs

Network equipment can generally implement similar IMA-protected functions to generate measurements (Claims) about the boot process of a device and enable corresponding remote attestation. Network Equipment Boot Logs combine the measurement and logging of boot components and operating system components (executables and files) into a single log file in a format identical to the IMA format. Note that the format used for logging measurement of boot components in this scheme differs from the boot logging strategy described

elsewhere in this document.

During the boot process of the network device, i.e., from BIOS to the end of the operating system and user-space, all files executed can be measured and logged in the order of their execution. When the Verifier initiates a remote attestation process (e.g., challenge-response remote attestation as defined in this document), the network equipment takes on the role of an Attester and can convey to the Verifier Claims that comprise the measurement log as well as the corresponding PCR values (Evidence) of a TPM.

The verifier can appraise the integrity (compliance with the Reference Values) of each executed file by comparing its measured value with the Reference Value. Based on the execution order, the Verifier can compute a PCR reference value (by replaying the log) and compare it to the Measurement Log Claims obtained in conjunction with the PCR Evidence to assess their trustworthiness with respect to an intended operational state.

Network equipment usually executes multiple components in parallel. This holds not only during the operating system loading phase, but also even during the BIOS boot phase. With this measurement log mechanism, network equipment can take on the role of an Attester, proving to the Verifier the trustworthiness of its boot process. Using the measurement log, Verifiers can precisely identify mismatching log entries to infer potentially tampered components.

This mechanism also supports scenarios that modify files on the Attester that are subsequently executed during the boot phase (e.g., updating/patching) by simply updating the appropriate Reference Values in Reference Integrity Manifests that inform Verifiers about how an Attester is composed.

Authors' Addresses

Henk Birkholz
Fraunhofer SIT
Rheinstrasse 75
64295 Darmstadt
Germany
Email: henk.birkholz@sit.fraunhofer.de

Michael Eckel
Fraunhofer SIT
Rheinstrasse 75
64295 Darmstadt
Germany

Email: michael.eckel@sit.fraunhofer.de

Shwetha Bhandari
ThoughtSpot
Email: shwetha.bhandari@thoughtspot.com

Eric Voit
Cisco Systems
Email: evoit@cisco.com

Bill Sulzen
Cisco Systems
Email: bsulzen@cisco.com

Liang Xia (Frank)
Huawei Technologies
101 Software Avenue, Yuhuatai District
Nanjing
Jiangsu, 210012
China
Email: Frank.Xialiang@huawei.com

Tom Laffey
Hewlett Packard Enterprise
Email: tom.laffey@hpe.com

Guy C. Fedorkow
Juniper Networks
10 Technology Park Drive
Westford
Email: gfedorkow@juniper.net