RATS Working Group
Internet-Draft
Intended status: Standards Track
Expires: 16 February 2026

E. Voit
Cisco
H. Birkholz
Fraunhofer SIT
T. Hardjono
MIT
T. Fossati
Linaro
V. Scarlata
Intel

15 August 2025

Attestation Results for Secure Interactions
draft-ietf-rats-ar4si-09

Abstract

This document defines reusable Attestation Result information elements. When these elements are offered to Relying Parties as Evidence, different aspects of Attester trustworthiness can be evaluated. Additionally, where the Relying Party is interfacing with a heterogeneous mix of Attesting Environment and Verifier types, consistent policies can be applied to subsequent information exchange between each Attester and the Relying Party.

This document also defines two serialisations of the proposed information model, utilising CBOR and JSON.

Discussion Venues

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

Source for this draft and an issue tracker can be found at https://github.com/ietf-rats-wg/draft-ietf-rats-ar4si.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

The first paragraph of the May 2021 US Presidential Executive Order on Improving the Nation's Cybersecurity [US-Executive-Order] ends with the statement "the trust we place in our digital infrastructure should be proportional to how trustworthy and transparent that infrastructure is." Later this order explores aspects of trustworthiness such as an auditable trust relationship, which it defines as an "agreed-upon relationship between two or more system elements that is governed by criteria for secure interaction, behavior, and outcomes."

The Remote ATtestation procedureS (RATS) architecture [RFC9334] provides a useful context for programmatically establishing and maintaining such auditable trust relationships. Specifically, the architecture defines conceptual messages conveyed between architectural subsystems to support trustworthiness appraisal. The RATS conceptual message used to convey evidence of trustworthiness is the Attestation Results. The Attestation Results includes Verifier generated appraisals of an Attester including such information as the identity of the Attester, the security mechanisms employed on this Attester, and the Attester's current state of trustworthiness.

Generated Attestation Results are ultimately conveyed to one or more Relying Parties. Reception of an Attestation Result enables a Relying Party to determine what action to take with regards to an

Attester. Frequently, this action will be to choose whether to allow the Attester to securely interact with the Relying Party over some connection between the two.

When determining whether to allow secure interactions with an Attester, a Relying Party is challenged with a number of difficult problems which it must be able to handle successfully. These problems include:

* What Attestation Results (AR) might a Relying Party be willing to trust from a specific Verifier?

* What information does a Relying Party need before allowing interactions or choosing policies to apply to a connection?

* What are the operating/environmental realities of the Attesting Environment where a Relying Party should only be able to associate a certain confidence regarding Attestation Results out of the Verifier? (In other words, different types of Trusted Execution Environments (TEE) need not be treated as equivalent.)

* How to make direct comparisons where there is a heterogeneous mix of Attesting Environments and Verifier types.

To address these problems, it is important that specific Attestation Result information elements are framed independently of Attesting Environment specific constraints. If they are not, a Relying Party would be forced to adapt to the syntax and semantics of many vendor specific environments. This is not a reasonable ask as there can be many types of Attesters interacting with or connecting to a Relying Party.

The business need therefore is for common Attestation Result information element definitions. With these definitions, consistent interaction or connectivity decisions can be made by a Relying Party where there is a heterogenous mix of Attesting Environment types and Verifier types.

This document defines information elements for Attestation Results in a way which normalizes the trustworthiness assertions that can be made from a diverse set of Attesters.

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.

1.2. Terminology

The following terms are imported from [RFC9334]: Appraisal Policy for Attestation Results, Attester, Attesting Environment, Claims, Evidence, Relying Party, Target Environment and Verifier.

[RFC9334] also describes topological patterns that illustrate the need for interoperable conceptual messages. The two patterns called "background-check model" and "passport model" are imported from the RATS architecture and used in this document as a reference to the architectural concepts: Background-Check Model and Passport Model.

Newly defined terms for this document:

AR-augmented Evidence: a bundle of Evidence which includes at least the following:

1. Verifier signed Attestation Results. These Attestation Results must include Identity Evidence for the Attester, a Trustworthiness Vector describing a Verifier’s most recent appraisal of an Attester, and some Verifier Proof-of-Freshness (PoF).

2. A Relying Party PoF which is bound to the Attestation Results of (1) by the Attester’s Attesting Environment signature.

3. Sufficient information to determine the elapsed interval between the Verifier PoF and Relying Party PoF.

Identity Evidence: Evidence which unambiguously identifies an identity. Identity Evidence could take different forms, such as a certificate, or a signature which can be appraised to have only been generated by a specific private/public key pair.

Trustworthiness Claim: a specific quanta of trustworthiness which can be assigned by a Verifier based on its appraisal policy.

Trustworthiness Tier: a categorization of the levels of

trustworthiness which may be assigned by a Verifier to a specific Trustworthiness Claim. These enumerated categories are: Affirming, Warning, Contraindicated, and None.

Trustworthiness Vector: a set of zero to many Trustworthiness Claims assigned during a single appraisal procedure by a Verifier using Evidence generated by an Attester. The vector is included within Attestation Results.

2. Attestation Results for Secure Interactions

A Verifier generates the Attestation Results used by a Relying Party. When a Relying Party needs to determine whether to permit communications with an Attester, these Attestation Results must contain a specific set of information elements. This section defines those information elements, and in some cases encodings for information elements.

2.1. Information driving a Relying Party Action

When the action is a communication establishment attempt with an Attester, there is only a limited set of actions which a Relying Party might take. These actions include:

* Allow or deny information exchange with the Attester. When there is a deny, reasons should be returned to the Attester.

* Establish a transport connection between an Attester and a specific context within a Relying Party (e.g., a TEE, or Virtual Routing Function (VRF).)

* Apply policies on this connection (e.g., rate limits).

There are three categories of information which must be conveyed to the Relying Party (which also is integrated with a Verifier) before it determines which of these actions to take.

1. Non-repudiable Identity Evidence Evidence which undoubtably identifies one or more entities involved with a communication.

2. Trustworthiness Claims Specifics a Verifier asserts with regards to its trustworthiness findings about an Attester.

3. Claim Freshness Establishes the time of last update (or refresh) of Trustworthiness Claims.

The following sections detail requirements for these three categories.

2.2. Non-repudiable Identity

Identity Evidence must be conveyed during the establishment of any trust-based relationship. Specific use cases will define the minimum types of identities required by a particular Relying Party as it evaluates Attestation Results, and perhaps additional associated Evidence. At a bare minimum, a Relying Party MUST start with the ability to verify the identity of a Verifier it chooses to trust. Attester identities may then be acquired through signed or encrypted communications with the Verifier identity and/or the pre-provisioning Attester public keys in the Attester.

During the Remote Attestation process, the Verifier’s identity must be established with a Relying Party, often via a Verifier signature across recent Attestation Results. This Verifier identity could only have come from a key pair maintained by a trusted developer or operator of the Verifier.

Additionally, each set of Attestation Results must be provably and non-reputably bound to the identity of the original Attesting Environment which was evaluated by the Verifier. This is accomplished via satisfying two requirements. First the Verifier signed Attestation Results MUST include sufficient Identity Evidence to ensure that this Attesting Environment signature refers to the same Attesting Environment appraised by the Verifier. Second, where the passport model is used as a subsystem, an Attesting Environment signature which spans the Verifier signature MUST also be included. As the Verifier signature already spans the Attester Identity as well as the Attestation Results, this restricts the viability of spoofing attacks.

In a subset of use cases, these two pieces of Identity Evidence may be sufficient for a Relying Party to successfully meet the criteria for its Appraisal Policy for Attestation Results. If the use case is a connection request, a Relying Party may simply then establish a transport session with an Attester after a successful appraisal. However an Appraisal Policy for Attestation Results will often be more nuanced, and the Relying Party may need additional information. Some Identity Evidence related policy questions which the Relying Party may consider include:

* Does the Relying Party only trust this Verifier to make Trustworthiness Claims on behalf a specific type of Attesting Environment? Might a mix of Verifiers be necessary to cover all mandatory Trustworthiness Claims?

* Does the Relying Party only accept connections from a verified-authentic software build from a specific software developer?

* Does the Relying Party only accept connections from specific preconfigured list of Attesters?

For any of these more nuanced appraisals, additional Identity Evidence or other policy related information must be conveyed or pre-provisioned during the formation of a trust context between the Relying Party, the Attester, the Attester's Attesting Environment, and the Verifier.

2.2.1. Attester and Attesting Environment

Per [RFC9334] Figure 2, an Attester and a corresponding Attesting Environment might not share common code or even hardware boundaries. Consequently, an Attester implementation needs to ensure that any Evidence which originates from outside the Attesting Environment MUST have been collected and delivered securely before any Attesting Environment signing may occur. After the Verifier performs its appraisal, it will include sufficient information in the Attestation Results to enable a Relying Party to have confidence that the Attester's trustworthiness is represented via Trustworthiness Claims signed by the appropriate Attesting Environment.

This document recognizes three general categories of Attesters.

1. HSM-based: A Hardware Security Module (HSM) based cryptoprocessor which hashes one or more streams of security measurements from an Attester within the Attesting Environment. Maintenance of this hash enables detection of an Attester which is not reporting the exact set of security measurements (such as log entries) taken within the Attesting Environment. An example of a HSM is a TPM2.0 [TPM2.0].

2. Process-based: An individual process which has its runtime memory encrypted by an Attesting Environment in a way that no other processes can read and decrypt that memory (e.g., [SGX] or [RFC9783].)

3. VM-based: An entire Guest VM (or a set of containers within a host) have been encrypted as a walled-garden unit by an Attesting Environment. The result is that the host operating system cannot read and decrypt what is executing within that VM (e.g., [SEV-SNP], [TDX] or [ARM-CCA].)

Each of these categories of Attesters above will be capable of generating Evidence which is protected using private keys / certificates which are not accessible outside of the corresponding Attesting Environment. The owner of these secrets is the owner of the identity which is bound within the Attesting Environment.

Effectively this means that for any Attester identity, there will exist a chain of trust ultimately bound to a hardware-based root of trust in the Attesting Environment. It is upon this root of trust that unique, non-repudiable Attester identities may be founded.

There are several types of Attester identities defined in this document. This list is extensible:

* chip-vendor: the vendor of the hardware chip used for the Attesting Environment (e.g., a primary Endorsement Key from a TPM)

* chip-hardware: specific hardware with specific firmware from an 'chip-vendor'

* target-environment: a unique instance of a software build running in an Attester (e.g., MRENCLAVE [SGX], an Identity Block [SEV-SNP], a Realm Initial Measurement (RIM) [ARM-CCA], or a hash which represents a set of software loaded since boot (e.g., TPM based integrity verification.))

* target-developer: the organizational unit responsible for a particular 'target-environment' (e.g., MRSIGNER [SGX])

* instance: a unique instantiated instance of an Attesting Environment running on 'chip-hardware' (e.g., an LDevID [IEEE802.1AR], an Instance ID [RFC9783] [ARM-CCA])

Based on the category of the Attesting Environment, different types of identities might be exposed by an Attester.

Attester Identity type	Process-based	VM-based	HSM-based
chip-vendor	Mandatory	Mandatory	Mandatory
chip-hardware	Mandatory	Mandatory	Mandatory
target-environment	Mandatory	Mandatory	Optional
target-developer	Mandatory	Optional	Optional
instance	Optional	Optional	Optional

Table 1

It is expected that drafts subsequent to this specification will provide the definitions and value domains for specific identities, each of which falling within the Attester identity types listed above. In some cases the actual unique identities might be encoded as complex structures. An example complex structure might be a 'target-environment' encoded as a Software Bill of Materials (SBOM).

With the identity definitions and value domains, a Relying Party will have sufficient information to ensure that the Attester identities and Trustworthiness Claims asserted are actually capable of being supported by the underlying type of Attesting Environment. Consequently, the Relying Party SHOULD require Identity Evidence which indicates of the type of Attesting Environment when it considers its Appraisal Policy for Attestation Results.

2.2.2. Verifier

For the Verifier identity, it is critical for a Relying Party to review the certificate and chain of trust for that Verifier. Additionally, the Relying Party must have confidence that the Trustworthiness Claims being relied upon from the Verifier considered the chain of trust for the Attesting Environment.

There are two categorizations of Verifier identities defined in this document:

* verifier build: a unique instance of a software build running as a Verifier.

* verifier developer: the organizational unit responsible for a particular 'verifier build'.

Within each category, communicating the identity can be accomplished via a variety of objects and encodings.

2.2.3. Communicating Identity

Any of the above identities used by the Appraisal Policy for Attestation Results needed to be pre-established by the Relying Party before, or provided during, the exchange of Attestation Results. When provided during this exchange, the identity may be communicated either implicitly or explicitly.

An example of explicit communication would be to include the following Identity Evidence directly within the Attestation Results: a unique identifier for an Attesting Environment, the name of a key which can be provably associated with that unique identifier, and the set of Attestation Results which are signed using that key. As these

Attestation Results are signed by the Verifier, it is the Verifier which is explicitly asserting the credentials it believes are trustworthy.

An example of implicit communication would be to include Identity Evidence in the form of a signature which has been placed over the Attestation Results asserted by a Verifier. It would be then up to the Relying Party's Appraisal Policy for Attestation Results to extract this signature and confirm that it only could have been generated by an Attesting Environment having access to a specific private key. This implicit identity communication is only viable if the Attesting Environment's public key is already known by the Relying Party.

One final step in communicating identity is proving the freshness of the Attestation Results to the degree needed by the Relying Party. A typical way to accomplish this is to include an element of freshness be embedded within a signed portion of the Attestation Results. This element of freshness reduces the identity spoofing risks from a replay attack. For more on this, see Section 2.4.

2.3. Trustworthiness Claims

2.3.1. Design Principles

Trust is not absolute. Trust is a belief in some aspect about an entity (in this case an Attester), and that this aspect is something which can be depended upon (in this case by a Relying Party.) Within the context of Remote Attestation, believability of this aspect is facilitated by a Verifier. This facilitation depends on the Verifier's ability to parse detailed Evidence from an Attester and then to assert conclusions about this aspect in a way interpretable by a Relying Party.

Specific aspects for which a Verifier will assert trustworthiness are defined in this section. These are known as Trustworthiness Claims. These claims have been designed to enable a common understanding between a broad array of Attesters, Verifiers, and Relying Parties. The following set of design principles have been applied in the Trustworthiness Claim definitions:

1. Expose a small number of Trustworthiness Claims.

Reason: a plethora of similar Trustworthiness Claims will result in divergent choices made on which to support between different Verifiers. This would place a lot of complexity in the Relying Party as it would be up to the Relying Party (and its policy language) to enable normalization across rich but incompatible Verifier object definitions.

2. Each Trustworthiness Claim enumerates only the specific states that could viably result in a different outcome after the Policy for Attestation Results has been applied.

   Reason: by explicitly disallowing the standardization of enumerated states which cannot easily be connected to a use case, we avoid forcing implementers from making incompatible guesses on what these states might mean.

3. Verifier and RP developers need explicit definitions of each state in order to accomplish the goals of (1) and (2).

   Reason: without such guidance, the Verifier will append plenty of raw supporting info. This relieves the Verifier of making the hard decisions. Of course, this raw info will be mostly non-interpretable and therefore non-actionable by the Relying Party.

4. Support standards and non-standard extensibility for (1) and (2).

   Reason: standard types of Verifier generated Trustworthiness Claims should be vetted by the full RATS working group, rather than being maintained in a repository which doesn't follow the RFC process. This will keep a tight lid on extensions which must be considered by the Relying Party's policy language. Because this process takes time, non-standard extensions will be needed for implementation speed and flexibility.

These design principles are important to keep the number of Verifier generated claims low, and to retain the complexity in the Verifier rather than the Relying Party.

2.3.2. Enumeration Encoding

Per design principle (2), each Trustworthiness Claim will only expose specific encoded values. To simplify the processing of these enumerations by the Relying Party, the enumeration will be encoded as a single signed 8 bit integer. These value assignments for this integer will be in four Trustworthiness Tiers which follow these guidelines:

None: The Verifier makes no assertions regarding this aspect of trustworthiness.

* Value 0: The Evidence received is insufficient to make a conclusion. Note: this should always be always treated equivalently by the Relying Party as no claim being made. I.e., the RP's Appraisal Policy for Attestation Results SHOULD NOT make any distinction between a Trustworthiness Claim with enumeration '0', and no Trustworthiness Claim being provided.

* Value 1: The Evidence received contains unknown elements which the Verifier is unable to evaluate. An example might be that the wrong type of Evidence has been delivered. Another case is that of Evidence coming from a composite Attester: a Verifier may understand only part of it and leave as "unknown" the Trustworthiness claims related to features it can't appraise.

* Value -1: A verifier malfunction occurred during the Verifier's appraisal processing.

Affirming: The Verifier affirms the Attester support for this aspect of trustworthiness.

* Values 2 to 31: A standards enumerated reason for affirming.

* Values -2 to -32: A non-standard reason for affirming.

Warning: The Verifier warns about this aspect of trustworthiness.

* Values 32 to 95: A standards enumerated reason for the warning.

* Values -33 to -96: A non-standard reason for the warning.

Contraindicated: The Verifier asserts the Attester is explicitly untrustworthy in regard to this aspect.

* Values 96 to 127: A standards enumerated reason for the contraindication.

* Values -97 to -128: A non-standard reason for the contraindication.

This enumerated encoding listed above will simplify the Appraisal Policy for Attestation Results. Such a policies may be as simple as saying that a specific Verifier has recently asserted Trustworthiness Claims, all of which are Affirming.

2.3.3. Assigning a Trustworthiness Claim value

In order to simplify design, only a single encoded value is asserted by a Verifier for any Trustworthiness Claim within a using the following process.

1. If applicable, a Verifier MUST assign a standardized value from the Contraindicated tier.

2. Else if applicable, a Verifier MUST assign a non-standardized value from the Contraindicated tier.

3. Else if applicable, a Verifier MUST assign a standardized value from the Warning tier.

4. Else if applicable, a Verifier MUST assign a non-standardized value from the Warning tier.

5. Else if applicable, a Verifier MUST assign a standardized value from the Affirming tier.

6. Else if applicable, a Verifier MUST assign a non-standardized value from the Affirming tier.

7. Else a Verifier MAY assign a 0 or -1.

2.3.4. Specific Claims

Following are the Trustworthiness Claims and their supported enumerations which may be asserted by a Verifier:

configuration: A Verifier has appraised an Attester’s configuration, and is able to make conclusions regarding the exposure of known vulnerabilities

- 0: No assertion

- 1: Evidence contains unknown elements which inhibit Verifer evaluation.

- -1: Verifier malfunction

- 2: The configuration is a known and approved config.

- 3: The configuration includes or exposes no known vulnerabilities.

- 32: The configuration includes or exposes known vulnerabilities.

36: Elements of the configuration relevant to security are unavailable to the Verifier.

96: The configuration is unsupportable as it exposes unacceptable security vulnerabilities.

99: Cryptographic validation of the Evidence has failed.

executables: A Verifier has appraised and evaluated relevant runtime files, scripts, and/or other objects which have been loaded into the Target environment's memory.

0: No assertion

1: Evidence contains unknown elements which inhibit Verifer evaluation.

-1: Verifier malfunction

2: Only a recognized genuine set of approved executables, scripts, files, and/or objects have been loaded during and after the boot process.

3: Only a recognized genuine set of approved executables have been loaded during the boot process.

32: Only a recognized genuine set of executables, scripts, files, and/or objects have been loaded. However the Verifier cannot vouch for a subset of these due to known bugs or other known vulnerabilities.

33: Runtime memory includes executables, scripts, files, and/or objects which are not recognized.

96: Runtime memory includes executables, scripts, files, and/or object which are contraindicated.

99: Cryptographic validation of the Evidence has failed.

file-system: A Verifier has evaluated a specific set of directories within the Attester's file system. (Note: the Verifier may or may not indicate what these directory and expected files are via an unspecified management interface.)

0: No assertion

1: Evidence contains unknown elements which inhibit Verifer evaluation.

-1: Verifier malfunction

2: Only a recognized set of approved files are found.

32: The file system includes unrecognized executables, scripts, or files.

96: The file system includes contraindicated executables, scripts, or files.

99: Cryptographic validation of the Evidence has failed.

hardware: A Verifier has appraised any Attester hardware and firmware which are able to expose fingerprints of their identity and running code.

0: No assertion

1: Evidence contains unknown elements which inhibit Verifer evaluation.

-1: Verifier malfunction

2: An Attester has passed its hardware and/or firmware verifications needed to demonstrate that these are genuine/ supported.

32: An Attester contains only genuine/supported hardware and/or firmware, but there are known security vulnerabilities.

96: Attester hardware and/or firmware is recognized, but its trustworthiness is contraindicated.

97: A Verifier does not recognize an Attester’s hardware or firmware, but it should be recognized.

99: Cryptographic validation of the Evidence has failed.

instance-identity: A Verifier has appraised an Attesting Environment’s unique identity based upon private key signed Evidence which can be correlated to a unique instantiated instance of the Attester. (Note: this Trustworthiness Claim should only be generated if the Verifier actually expects to recognize the unique identity of the Attester.)

0: No assertion

1: Evidence contains unknown elements which inhibit Verifer

evaluation.

-1: Verifier malfunction

2: The Attesting Environment is recognized, and the associated instance of the Attester is not known to be compromised.

96: The Attesting Environment is recognized, and but its unique private key indicates a device which is not trustworthy.

97: The Attesting Environment is not recognized; however the Verifier believes it should be.

99: Cryptographic validation of the Evidence has failed.

runtime-opaque: A Verifier has appraised the visibility of Attester objects in memory from perspectives outside the Attester.

0: No assertion

1: Evidence contains unknown elements which inhibit Verifer evaluation.

-1: Verifier malfunction

2: the Attester’s executing Target Environment and Attesting Environments are encrypted and within Trusted Execution Environment(s) opaque to the operating system, virtual machine manager, and peer applications. (Note: This value corresponds to the protections asserted by O.RUNTIME_CONFIDENTIALITY from [GP-TEE-PP])

32: the Attester’s executing Target Environment and Attesting Environments inaccessible from any other parallel application or Guest VM running on the Attester’s physical device. (Note that unlike "1" these environments are not encrypted in a way which restricts the Attester’s root operator visibility. See O.TA_ISOLATION from [GP-TEE-PP].)

96: The Verifier has concluded that in memory objects are unacceptably visible within the physical host that supports the Attester.

99: Cryptographic validation of the Evidence has failed.

sourced-data: A Verifier has evaluated of the integrity of data objects from external systems used by the Attester.

0: No assertion

1: Evidence contains unknown elements which inhibit Verifer evaluation.

-1: Verifier malfunction

2: All essential Attester source data objects have been provided by other Attester(s) whose most recent appraisal(s) had both no Trustworthiness Claims of "0" where the current Trustworthiness Claim is "Affirming", as well as no "Warning" or "Contraindicated" Trustworthiness Claims.

32: Attester source data objects come from unattested sources, or attested sources with "Warning" type Trustworthiness Claims.

96: Attester source data objects come from contraindicated sources.

99: Cryptographic validation of the Evidence has failed.

storage-opaque: A Verifier has appraised that an Attester is capable of encrypting persistent storage. (Note: Protections must meet the capabilities of [OMTP-ATE] Section 5, but need not be hardware tamper resistant.)

0: No assertion

1: Evidence contains unknown elements which inhibit Verifer evaluation.

-1: Verifier malfunction

2: the Attester encrypts all secrets in persistent storage via using keys which are never visible outside an HSM or the Trusted Execution Environment hardware.

32: the Attester encrypts all persistently stored secrets, but without using hardware backed keys

96: There are persistent secrets which are stored unencrypted in an Attester.

99: Cryptographic validation of the Evidence has failed.

It is possible for additonal Trustworthiness Claims and enumerated values to be defined in subsequent documents. At the same time, the standardized Trustworthiness Claim values listed above have been

designed so there is no overlap within a Trustworthiness Tier. As a result, it is possible to imagine a future where overlapping Trustworthiness Claims within a single Trustworthiness Tier may be defined. Wherever possible, the Verifier SHOULD assign the best fitting standardized value.

Where a Relying Party doesn't know how to handle a particular Trustworthiness Claim, it MAY choose an appropriate action based on the Trustworthiness Tier under which the enumerated value fits.

It is up to the Verifier to publish the types of evaluations it performs when determining how Trustworthiness Claims are derived for a type of any particular type of Attester. It is out of the scope of this document for the Verifier to provide proof or specific logic on how a particular Trustworthiness Claim which it is asserting was derived.

2.3.5. Trustworthiness Vector

Multiple Trustworthiness Claims may be asserted about an Attesting Environment at single point in time. The set of Trustworthiness Claims inserted into an instance of Attestation Results by a Verifier is known as a Trustworthiness Vector. The order of Claims in the vector is NOT meaningful. A Trustworthiness Vector with no Trustworthiness Claims (i.e., a null Trustworthiness Vector) is a valid construct. In this case, the Verifier is making no Trustworthiness Claims but is confirming that an appraisal has been made.

2.3.6. Trustworthiness Vector for a type of Attesting Environment

Some Trustworthiness Claims are implicit based on the underlying type of Attesting Environment. For example, a validated MRSIGNER identity can be present where the underlying [SGX] hardware is authentic. Where such implicit Trustworthiness Claims exist, they do not have to be explicitly included in the Trustworthiness Vector. However, these implicit Trustworthiness Claims SHOULD be considered as being present by the Relying Party. Another way of saying this is if a Trustworthiness Claim is automatically supported as a result of coming from a specific type of TEE, that claim need not be redundantly articulated. Such implicit Trustworthiness Claims can be seen in the tables within Appendix B.2 and Appendix B.3.

Additionally, there are some Trustworthiness Claims which cannot be adequately supported by an Attesting Environment. For example, it would be difficult for an Attester that includes only a TPM (and no other TEE) from ever having a Verifier appraise support for 'runtime-opaque'. As such, a Relying Party would be acting properly if it rejects any non-supportable Trustworthiness Claims asserted from a Verifier.

As a result, the need for the ability to carry a specific Trustworthiness Claim will vary by the type of Attesting Environment. Example mappings can be seen in Appendix B.

2.4. Freshness

A Relying Party will care about the recentness of the Attestation Results, and the specific Trustworthiness Claims which are embedded. All freshness mechanisms of [RFC9334], Section 10 are supportable by this specification.

Additionally, a Relying Party may track when a Verifier expires its confidence for the Trustworthiness Claims or the Trustworthiness Vector as a whole. Mechanisms for such expiry are not defined within this document.

There is a subset of secure interactions where the freshness of Trustworthiness Claims may need to be revisited asynchronously. This subset is when trustworthiness depends on the continuous availability of a transport session between the Attester and Relying Party. With such connectivity dependent Attestation Results, if there is a reboot which resets transport connectivity, all established Trustworthiness Claims should be cleared. Subsequent connection re-establishment will allow fresh new Trustworthiness Claims to be delivered.

3. Data Model

The following CDDL [RFC8610] defines the necessary AR4SI types for use in CBOR and JSON serializations.

Other serializations are possible but must be defined in subsequent documents.

3.1. Trustworthiness Vector

The trustworthiness-vector is defined as follows:

```
trustworthiness-vector = non-empty<{
  ? instance-identity-label => trustworthiness-claim
  ? configuration-label => trustworthiness-claim
  ? executables-label => trustworthiness-claim
  ? file-system-label => trustworthiness-claim
  ? hardware-label => trustworthiness-claim
  ? runtime-opaque-label => trustworthiness-claim
  ? storage-opaque-label => trustworthiness-claim
  ? sourced-data-label => trustworthiness-claim
}>

instance-identity-label = JC<"instance-identity", 0>
configuration-label = JC<"configuration", 1>
executables-label = JC<"executables", 2>
file-system-label = JC<"file-system", 3>
hardware-label = JC<"hardware", 4>
runtime-opaque-label = JC<"runtime-opaque", 5>
storage-opaque-label = JC<"storage-opaque", 6>
sourced-data-label = JC<"sourced-data", 7>

trustworthiness-claim = -128..127
```

Figure 1: Trustworthiness Vector

This type contains an entry for each one of the eight AR4SI appraisals that have been conducted on the submitted evidence (Section 2.3.4).

The value of each entry is chosen in the -128..127 range according to the rules described in Section 2.3.3 and Section 2.3.4.

All categories are optional.

A missing entry means that the verifier makes no claim about this specific appraisal facet because the category is not applicable to the submitted evidence.

As required by the non-empty macro, at least one entry MUST be present in the vector.

3.2. Trustworthiness Tiers

The trustworthiness-tier type represents one of the equivalency classes in which the trustworthiness-claim space is partitioned.

See Section 2.3.2 for the details.

The allowed values for the type are as follows:

```
trustworthiness-tier-none-label = JC<"none", 0>
trustworthiness-tier-affirming-label = JC<"affirming", 2>
trustworthiness-tier-warning-label = JC<"warning", 32>
trustworthiness-tier-contraindicated-label = JC<"contraindicated", 96>

trustworthiness-tier /= trustworthiness-tier-none-label
trustworthiness-tier /= trustworthiness-tier-affirming-label
trustworthiness-tier /= trustworthiness-tier-warning-label
trustworthiness-tier /= trustworthiness-tier-contraindicated-label
```

Figure 2: Trustworthiness Tiers

3.3. Verifier ID

The verifier-id type identifies the software that runs the verifier according to the information model defined in Section 2.2.2.

```
verifier-id = {
  developer-label => text
  build-label => text
}

developer-label = JC<"developer", 0>
build-label = JC<"build", 1>
```

Figure 3: Verifier ID

The fields are:

build (mandatory) A text string that uniquely identifies the software build running the verifier.

developer (mandatory) A text string that uniquely identifies the organizational unit responsible for this build.

3.4. Consolitated CDDL

Figure 4 contains the CDDL of the entire AR4SI type system.

```
=============== NOTE: '\' line wrapping per RFC 8792 ================

$.start.$ = trustworthiness-vector / trustworthiness-tier / verifier\
                                              -id / trustworthiness-claim
trustworthiness-vector = non-empty<{
    ? instance-identity-label => trustworthiness-claim,
    ? configuration-label => trustworthiness-claim,
    ? executables-label => trustworthiness-claim,
    ? file-system-label => trustworthiness-claim,
    ? hardware-label => trustworthiness-claim,
    ? runtime-opaque-label => trustworthiness-claim,
    ? storage-opaque-label => trustworthiness-claim,
    ? sourced-data-label => trustworthiness-claim,
}>
instance-identity-label = JC<"instance-identity", 0>
configuration-label = JC<"configuration", 1>
executables-label = JC<"executables", 2>
file-system-label = JC<"file-system", 3>
hardware-label = JC<"hardware", 4>
runtime-opaque-label = JC<"runtime-opaque", 5>
storage-opaque-label = JC<"storage-opaque", 6>
sourced-data-label = JC<"sourced-data", 7>
trustworthiness-claim = -128 .. 127
trustworthiness-tier-none-label = JC<"none", 0>
trustworthiness-tier-affirming-label = JC<"affirming", 2>
trustworthiness-tier-warning-label = JC<"warning", 32>
trustworthiness-tier-contraindicated-label = JC<"contraindicated", \
                                                                    96>
trustworthiness-tier /= trustworthiness-tier-none-label / \
trustworthiness-tier-affirming-label / trustworthiness-tier-warning-\
                    label / trustworthiness-tier-contraindicated-label
verifier-id = {
  developer-label => text,
  build-label => text,
}
developer-label = JC<"developer", 0>
build-label = JC<"build", 1>
non-empty<M> = M .within ({+ any => any})
JSON-ONLY<J> = J .feature "json"
CBOR-ONLY<C> = C .feature "cbor"
JC<J, C> = JSON-ONLY<J> / CBOR-ONLY<C>
```

Figure 4: AR4SI CDDL

4. Secure Interactions Models

There are multiple ways of providing a Trustworthiness Vector to a Relying Party. This section describes two alternatives.

4.1. Background-Check

4.1.1. Verifier Retrieval

It is possible to for a Relying Party to follow the Background-Check Model defined in Section 5.2 of [RFC9334]. In this case, a Relying Party will receive Attestation Results containing the Trustworthiness Vector directly from a Verifier. These Attestation Results can then be used by the Relying Party in determining the appropriate treatment for interactions with the Attester.

While applicable in some cases, the utilization of the Background-Check Model without modification has potential drawbacks in other cases. These include:

* Verifier scale: if the Attester has many Relying Parties, a Verifier appraising that Attester could be frequently be queried based on the same Evidence.

* Information leak: Evidence which the Attester might consider private can be visible to the Relying Party. Hiding that Evidence could devalue any resulting appraisal.

* Latency: a Relying Party will need to wait for the Verifier to return Attestation Results before proceeding with secure interactions with the Attester.

An implementer should examine these potential drawbacks before selecting this alternative.

4.1.2. Co-resident Verifier

A simplified Background-Check Model may exist in a very specific case. This is where the Relying Party and Verifier functions are co-resident. This model is appropriate when:

* Some hardware-based private key is used by an Attester while proving its identity as part of a mutually authenticated secure channel establishment with the Relying Party, and

* this Attester identity is accepted as sufficient proof of Attester integrity.

Effectively this means that detailed forensic capabilities of a robust Verifier are unnecessary because it is accepted that the code and operational behavior of the Attester cannot be manipulated after TEE initialization.

An example of such a scenario may be when an SGX's MRENCLAVE and MRSIGNER values have been associated with a known QUOTE value. And the code running within the TEE is not modifiable after launch.

4.2. Below Zero Trust

Zero Trust Architectures are referenced in [US-Executive-Order] eleven times. However despite this high profile, there is an architectural gap with Zero Trust. The credentials used for authentication and admission control can be manipulated on the endpoint. Attestation can fill this gap through the generation of a compound credential called AR-augmented Evidence. This compound credential is rooted in the hardware based Attesting Environment of an endpoint, plus the trustworthiness of a Verifier. The overall solution is known as "Below Zero Trust" as the compound credential cannot be manipulated or spoofed by an administrator of an endpoint with root access. This solution is not adversely impacted by the potential drawbacks with pure background-check described above.

To kick-off the "Below Zero Trust" compound credential creation sequence, a Verifier evaluates an Attester and returns signed Attestation Results back to this original Attester no less frequently than a well-known interval. This interval may also be asynchronous, based on the changing of certain Evidence as described in [I-D.ietf-rats-network-device-subscription].

When a Relying Party is to receive information about the Attester's trustworthiness, the Attesting Environment assembles the minimal set of Evidence which can be used to confirm or refute whether the Attester remains in the state of trustworthiness represented by the AR. To this Evidence, the Attesting Environment appends the signature from the most recent AR as well as a Relying Party Proof-of-Freshness. The Attesting Environment then signs the combination.

The Attester then assembles AR Augmented Evidence by taking the signed combination and appending the full AR. The assembly now consists of two independent but semantically bound sets of signed Evidence.

The AR Augmented Evidence is then sent to the Relying Party. The Relying Party then can appraise these semantically bound sets of signed Evidence by applying an Appraisal Policy for Attestation Results as described below. This policy will consider both the AR as well as additional information about the Attester within the AR Augmented Evidence when determining what action to take.

This alternative combines the [RFC9334] Sections 5.1 Passport Model and Section 5.2 Background-Check Model. Figure 5 describes this flow of information. The flows within this combined model are mapped to [RFC9334] in the following way. "Verifier A" below corresponds to the "Verifier" Figure 5 within [RFC9334]. And "Relying Party/ Verifier B" below corresponds to the union of the "Relying Party" and "Verifier" boxes within Figure 6 of [RFC9334]. This union is possible because Verifier B can be implemented as a simple, self-contained process. The resulting combined process can appraise the AR-augmented Evidence to determine whether an Attester qualifies for secure interactions with the Relying Party. The specific steps of this process are defined later in this section.

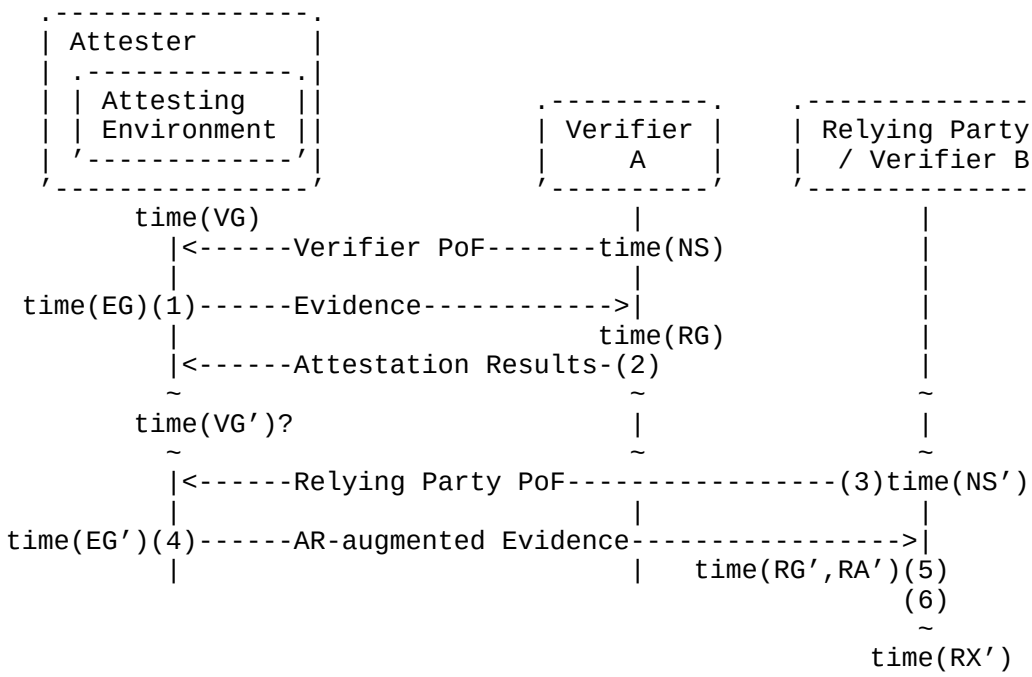

Figure 5: Below Zero Trust

The interaction model depicted above includes specific time related events from Appendix A of [RFC9334]. With the identification of these time related events, time duration/interval tracking becomes possible. Such duration/interval tracking can become important if the Relying Party cares if too much time has elapsed between the Verifier PoF and Relying Party PoF. If too much time has elapsed, perhaps the Attestation Results themselves are no longer trustworthy.

Note that while time intervals will often be relevant, there is a simplified case that does not require a Relying Party's PoF in step (3). In this simplified case, the Relying Party trusts that the Attester cannot be meaningfully changed from the outside during any reportable interval. Based on that assumption, and when this is the case then the step of the Relying Party PoF can be safely omitted.

In all cases, appraisal policies define the conditions and prerequisites for when an Attester does qualify for secure interactions. To qualify, an Attester has to be able to provide all of the mandatory affirming Trustworthiness Claims and identities needed by a Relying Party's Appraisal Policy for Attestation Results, and none of the disqualifying detracting Trustworthiness Claims.

More details on each interaction step of Below Zero Trust are as follows. The numbers used in this sequence match to the numbered steps in Figure 5:

1. An Attester sends Evidence which is provably fresh to Verifier A at time(EG). Freshness from the perspective of Verifier A MAY be established with Verifier PoF such as a nonce.

2. Verifier A appraises (1), then sends the following items back to that Attester within Attestation Results:

    1. the verified identity of the Attesting Environment,

    2. the Verifier A appraised Trustworthiness Vector of an Attester,

    3. a freshness proof associated with the Attestation Results,

    4. a Verifier signature across (2.1) though (2.3).

3. At time(EG') a Relying Party PoF (such as a nonce) known to the Relying Party is sent to the Attester.

4. The Attester generates and sends AR-augmented Evidence to the Relying Party/Verifier B. This AR-augmented Evidence includes:

    1. The Attestation Results from (2)

    2. Any (optionally) new incremental Evidence from the Attesting Environment

3. Attestation Environment signature which spans a hash of the Attestation Results (such as the signature of (2.4)), the proof-of-freshness from (3), and (4.2). Note: this construct allows the delta of time between (2.3) and (3) to be definitively calculated by the Relying Party.

5. On receipt of (4), the Relying Party applies its Appraisal Policy for Attestation Results. At minimum, this appraisal policy process must include the following:

    1. Verify that (4.3) includes the nonce from (3).

    2. Use a local certificate to validate the signature (4.1).

    3. Verify that the hash from (4.3) matches (4.1)

    4. Use the identity of (2.1) to validate the signature of (4.3).

    5. Failure of any steps (5.1) through (5.4) means the link does not meet minimum validation criteria, therefore appraise the link as having a null Verifier B Trustworthiness Vector. Jump to step (6.1).

    6. When there is large or uncertain time gap between time(EG) and time(EG’), the link should be assigned a null Verifier B Trustworthiness Vector. Jump to step (6.1).

    7. Assemble the Verifier B Trustworthiness Vector

        1. Copy Verifier A Trustworthiness Vector to Verifier B Trustworthiness Vector

        2. Add implicit Trustworthiness Claims inherent to the type of TEE.

        3. Prune any Trustworthiness Claims unsupportable by the Attesting Environment.

        4. Prune any Trustworthiness Claims the Relying Party doesn’t accept from this Verifier.

6. The Relying Party takes action based on Verifier B’s appraised Trustworthiness Vector, and applies the Appraisal Policy for Attestation Results. Following is a reasonable process for such evaluation:

1. Prune any Trustworthiness Claims from the Trustworthiness Vector not used in the Appraisal Policy for Attestation Results.

2. Allow the information exchange from the Attester into a Relying Party context in the Appraisal Policy for Attestation Results where the Verifier B appraised Trustworthiness Vector includes all the mandatory Trustworthiness Claims are in the "Affirming" value range, and none of the disqualifying Trustworthiness Claims are in the "Contraindicated" value range.

3. Disallow any information exchange into a Relying Party context for which that Verifier B appraised Trustworthiness Vector is not qualified.

As link layer protocols re-authenticate, steps (1) to (2) and steps (3) to (6) will independently refresh. This allows the Trustworthiness of Attester to be continuously re-appraised. There are only specific event triggers which will drive the refresh of Evidence generation (1), Attestation Result generation (2), or AR-augmented Evidence generation (4):

* life-cycle events, e.g. a change to an Authentication Secret of the Attester or an update of a software component.

* uptime-cycle events, e.g. a hard reset or a re-initialization of an Attester.

* authentication-cycle events, e.g. a link-layer interface reset could result in a new (4).

4.3. Mutual Attestation

In the interaction models described above, each device on either side of a secure interaction may require remote attestation of its peer. This process is known as mutual-attestation. To support mutual-attestation, the interaction models listed above may be run independently on either side of the connection.

4.4. Transport Protocol Integration

Either unidirectional attestation or mutual attestation may be supported within the protocol interactions needed for the establishment of a single transport session. While this document does not mandate specific transport protocols, messages containing the Attestation Results and AR Augmented Evidence can be passed within an authentication framework such the EAP protocol [RFC5247]

over TLS [RFC8446].

5. Privacy Considerations

Privacy Considerations Text

6. Security Considerations

Security Considerations Text

7. IANA Considerations

See Body.

8. References

8.1. Normative References

[GP-TEE-PP] "Global Platform TEE Protection Profile v1.3", September 2020, <https://globalplatform.org/specs-library/tee-protection-profile-v1-3/>.

[OMTP-ATE] "Open Mobile Terminal Platform - Advanced Trusted Environment", May 2009, <https://www.gsma.com/newsroom/wp-content/uploads/2012/03/omtpadvancedtrustedenvironmentomtptr1v11.pdf>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/rfc/rfc2119>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/rfc/rfc8174>.

[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/rfc/rfc8610>.

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

8.2. Informative References

[ARM-CCA] Frost, S., Fossati, T., and G. Mandyam, "Arm's Confidential Compute Architecture Reference Attestation Token", Work in Progress, Internet-Draft, draft-ffm-rats-cca-token-01, 3 March 2025, <https://datatracker.ietf.org/doc/html/draft-ffm-rats-cca-token-01>.

[I-D.ietf-rats-network-device-subscription] Birkholz, H., Voit, E., and W. Pan, "Attestation Event Stream Subscription", Work in Progress, Internet-Draft, draft-ietf-rats-network-device-subscription-07, 21 July 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-rats-network-device-subscription-07>.

[IEEE802.1AR] "802.1AR: Secure Device Identity", 2 August 2018, <https://ieeexplore.ieee.org/document/8423794>.

[RFC5247] Aboba, B., Simon, D., and P. Eronen, "Extensible Authentication Protocol (EAP) Key Management Framework", RFC 5247, DOI 10.17487/RFC5247, August 2008, <https://www.rfc-editor.org/rfc/rfc5247>.

[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/rfc/rfc8446>.

[RFC9783] Tschofenig, H., Frost, S., Brossard, M., Shaw, A., and T. Fossati, "Arm's Platform Security Architecture (PSA) Attestation Token", RFC 9783, DOI 10.17487/RFC9783, June 2025, <https://www.rfc-editor.org/rfc/rfc9783>.

[SEV-SNP] "AMD SEV-SNP: Stregthening VM Isolation with Integrity Protection and More", 2020, <https://www.amd.com/system/files/TechDocs/SEV-SNP-strengthening-vm-isolation-with-integrity-protection-and-more.pdf>.

[SGX] "Supporting Third Party Attestation for Intel SGX with Intel Data Center Attestation Primitives", 2017, <https://software.intel.com/content/dam/develop/external/us/en/documents/intel-sgx-support-for-third-party-attestation-801017.pdf>.

[TDX] "Intel Trust Domain Extensions", 2020, <https://software.intel.com/content/dam/develop/external/us/en/documents/tdx-whitepaper-final9-17.pdf>.

[TPM-ID] "TPM Keys for Platform Identity for TPM 1.2", August 2015, <https://www.trustedcomputinggroup.org/wp-content/uploads/TPM_Keys_for_Platform_Identity_v1_0_r3_Final.pdf>.

[TPM2.0] "Trusted Platform Module Library - Part 1: Architecture", n.d., <https://trustedcomputinggroup.org/wp-content/uploads/TPM-Rev-2.0-Part-1-Architecture-01.07-2014-03-13.pdf>.

[US-Executive-Order] "Executive Order on Improving the Nation’s Cybersecurity", 12 May 2021, <https://bidenwhitehouse.archives.gov/briefing-room/presidential-actions/2021/05/12/executive-order-on-improving-the-nations-cybersecurity/>.

Appendix A. Implementation Guidance

A.1. Supplementing Trustworthiness Claims

What has been encoded into each Trustworthiness Claim is the domain of integer values which is likely to drive a different programmatic decision in the Relying Party’s Appraisal Policy for Attestation Results. This will not be the only thing a Relying Party’s Operations team might care to track for measurement or debugging purposes.

There is also the opportunity for the Verifier to include supplementary Evidence beyond a set of asserted Trustworthiness Claims. It is recommended that if supplementary Evidence is provided by the Verifier within the Attestation Results, that this supplementary Evidence includes a reference to a specific Trustworthiness Claim. This will allow a deeper understanding of some of the reasoning behind the integer value assigned.

Appendix B. Supportable Trustworthiness Claims

The following is a table which shows what Claims are supportable by different Attesting Environment types. Note that claims MAY BE implicit to an Attesting Environment type, and therefore do not have to be included in the Trustworthiness Vector to be considered as set by the Relying Party.

B.1. Supportable Trustworthiness Claims for HSM-based CC

Following are Trustworthiness Claims which MAY be set for a HSM-based Confidential Computing Attester. (Such as a TPM [TPM-ID].)

Trustworthiness Claim	Required?	Appraisal Method
configuration	Optional	Verifier evaluation of Attester reveals no configuration lines which expose the Attester to known security vulnerabilities. This may be done with or without the involvement of a TPM PCR.
executables	Yes	Checks the TPM PCRs for the static operating system, and for any tracked files subsequently loaded
file-system	No	Can be supported, but TPM tracking is unlikely
hardware	Yes	If TPM PCR check ok from BIOS checks, through Master Boot Record configuration
instance-identity	Optional	Check IDevID
runtime-opaque	n/a	TPMs are not recommended to provide a sufficient technology base for this Trustworthiness Claim.
sourced-data	n/a	TPMs are not recommended to provide a sufficient technology base for this Trustworthiness Claim.
storage-opaque	Minimal	With a TPM, secure storage space exists and is writeable by external applications. But the space is so limited that it often is used just be used to store keys.

Table 2

Setting the Trustworthiness Claims may follow the following logic at the Verifier A within (2) of Figure 5:

```
   Start: Evidence received starts the generation of a new
   Trustworthiness Vector.  (e.g.,  TPM Quote Received, log received,
   or appraisal timer expired)

   Step 0: set Trustworthiness Vector = Null

   Step 1: Is there sufficient fresh signed evidence to appraise?
     (yes) - No Action
     (no) -  Goto Step 6

   Step 2: Appraise Hardware Integrity PCRs
      if (hardware NOT "0") - push onto vector
      if (hardware NOT affirming or warning), go to Step 6

   Step 3: Appraise Attesting Environment identity
      if (instance-identity <> "0") - push onto vector

   Step 4: Appraise executable loaded and filesystem integrity
      if (executables NOT "0") - push onto vector
      if (executables NOT affirming or warning), go to Step 6

   Step 5: Appraise all remaining Trustworthiness Claims
           Independently and set as appropriate.

   Step 6: Assemble Attestation Results, and push to Attester

   End
```

B.2. Supportable Trustworthiness Claims for process-based CC

Following are Trustworthiness Claims which MAY be set for a process-based Confidential Computing based Attester. (Such as a SGX Enclaves and TrustZone.)

Trustworthiness Claim	Required?	Appraisal Method
instance-identity	Optional	Internally available in TEE. But keys might not be known/ exposed to the Relying Party by the Attesting Environment.
configuration	Optional	If done, this is at the Application Layer. Plus each process needs it own protection mechanism as the protection is limited to the process itself.
executables	Optional	Internally available in TEE. But keys might not be known/ exposed to the Relying Party by the Attesting Environment.
file-system	Optional	Can be supported by application, but process-based CC is not a sufficient technology base for this Trustworthiness Claim.
hardware	Implicit in signature	At least the TEE is protected here. Other elements of the system outside of the TEE might need additional protections is used by the application process.
runtime-opaque	Implicit in signature	From the TEE
storage-opaque	Implicit in signature	Although the application must assert that this function is used by the code itself.
sourced-data	Optional	Will need to be supported by application code

Table 3

B.3. Supportable Trustworthiness Claims for VM-based CC

Following are Trustworthiness Claims which MAY be set for a VM-based Confidential Computing based Attester. (Such as SEV, TDX, ACCA, SEV-SNP.)

Trustworthiness Claim	Required?	Appraisal Method
instance-identity	Optional	Internally available in TEE. But keys might not be known/ exposed to the Relying Party by the Attesting Environment.
configuration	Optional	Requires application integration. Easier than with process-based solution, as the whole protected machine can be evaluated.
executables	Optional	Internally available in TEE. But keys might not be known/ exposed to the Relying Party by the Attesting Environment.
file-system	Optional	Can be supported by application
hardware	Chip dependent	At least the TEE is protected here. Other elements of the system outside of the TEE might need additional protections is used by the application process.
runtime-opaque	Implicit in signature	From the TEE
storage-opaque	Chip dependent	Although the application must assert that this function is used by the code itself.
sourced-data	Optional	Will need to be supported by application code

Table 4

Appendix C. Some issues being worked

It is possible for a cluster/hierarchy of Verifiers to have aggregate AR which are perhaps signed/endorsed by a lead Verifier. What should be the Proof-of-Freshness or Verifier associated with any of the aggregate set of Trustworthiness Claims?

There will need to be a subsequent document which documents how these objects which will be translated into a protocol on a wire (e.g. EAP on TLS). Some breakpoint between what is in this draft, and what is in specific drafts for wire encoding will need to be determined. Questions like architecting the cluster/hierarchy of Verifiers fall into this breakdown.

For some Trustworthiness Claims, there could be value in identifying a specific Appraisal Policy for Attestation Results applied within the Attester. One way this could be done would be a URI which identifies the policy used at Verifier A, and this URI would reference a specific Trustworthiness Claim. As the URI also could encode the version of the software, it might also act as a mechanism to signal the Relying Party to refresh/re-evaluate its view of Verifier A. Do we need this type of structure to be included here? Should it be in subsequent documents?

Expand the variant of Figure 5 which requires no Relying Party PoF into its own picture.

In what document (if any) do we attempt normalization of the identity claims between different types of TEE. E.g., does MRSIGNER plus extra loaded software = the sum of TrustZone Signer IDs for loaded components?

Appendix D. Contributors

Guy Fedorkow

Email: gfedorkow@juniper.net

Dave Thaler

Email: dthaler@microsoft.com

Ned Smith

Email: ned.smith@intel.com

Lawrence Lundblade

Email: lgl@island-resort.com

Authors' Addresses

Eric Voit
Cisco Systems
Email: evoit@cisco.com

Henk Birkholz
Fraunhofer SIT
Rheinstrasse 75
64295 Darmstadt
Germany
Email: henk.birkholz@sit.fraunhofer.de

Thomas Hardjono
MIT
Email: hardjono@mit.edu

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

Vincent Scarlata
Intel
Email: vincent.r.scarlata@intel.com