Remote ATtestation ProcedureS
Internet-Draft
Intended status: Standards Track
Expires: 2 October 2026

A. Poulain
A. Kaci
Secure-IC

31 March 2026

Attestation of Hardware Components
draft-paka-rats-hardware-component-attestation-00

Abstract

Hardware components constitute the foundation of all computations and therefore play a critical role in system integrity and reliability. Existing attestation mechanisms primarily rely on manufacturer endorsements, which provide limited visibility into the runtime behavior of hardware. This document extends the Remote ATtestation procedureS (RATS) architecture by defining a data model and guidelines for including measurements of hardware components in attestation Evidence. These measurements may represent physical properties, results of self-tests, or behavioral observations. The document considers a threat model that includes both adversarial actions and physical phenomena such as environmental variations and aging. It proposes abstract interfaces for collecting measurements, enabling interoperability while remaining agnostic to implementation mechanisms, and outlines a security model for their use in appraisal.

About This Document

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

The latest revision of this draft can be found at https://github.com/antoinepoulain/hardware-component-attestation/blob/main/draft-paka-rats-hardware-component-attestation.txt. Status information for this document may be found at https://datatracker.ietf.org/doc/draft-paka-rats-hardware-component-attestation/.

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

Source for this draft and an issue tracker can be found at https://github.com/antoinepoulain/hardware-component-attestation.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

Hardware components form the foundation upon which all computations rely. Therefore, the correctness and integrity of software execution depend on the proper functioning of the underlying hardware, which can be considered a root of trust for computation.

Modern systems increasingly adopt disaggregated architectures, such as chiplet-based designs and large-scale heterogeneous platforms. These systems integrate hardware components from multiple sources, introducing new attack surfaces.

At the same time, zero trust principles encourage reducing reliance on static trust anchors and Endorsements in favor of Evidence reflecting the actual runtime state of components, consequently enabling more dependable assessment of both security and safety properties. However, current attestation mechanisms for hardware components primarily rely on manufacturer-issued Endorsements, which capture properties established prior to deployment but provide limited visibility into runtime hardware behavior.

This document considers a threat model in which hardware components may be affected not only by adversarial actions, but also by physical phenomena such as environmental variations, aging, and natural degradation (see Section 8). This wider scope is motivated by the fact that hardware components, are directly influenced by physical conditions that can alter their behavior over time or under stress. Therefore, assessing the runtime state of hardware requires taking into account both intentional attacks and non-adversarial effects that may lead to faults or degraded operation. These aspects are particularly important in systems with strong safety and reliability requirements.

To address these limitations, this document defines a data model and provides guidelines for including hardware component measurements in attestation Evidence (Section 5.3), as described in the RATS architecture [RFC9334]. By incorporating runtime hardware measurements, attestation can provide improved visibility into the integrity and reliability of systems. This document also outlines a security model for such measurements and provides examples of existing technologies that can be leveraged to obtain them (Section 7). These examples are informational only and do not mandate specific implementations. Instead, this document remains agnostic to the underlying measurement mechanisms and focuses on defining abstract interfaces (Section 4.1) and a data model for obtaining and representing such measurements.

2. Terminology

2.1. 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.

2.2. Definitions

The terminology defined in [RFC9334] is reused throughout this document. Some definitions from RATS specifications are refined here to fit the context presented in this document.

* Measurement Unit: A hardware mechanism (a circuit) or software logic with the ability to measure a hardware component. Software logic used to trigger a measurement is not considered a Measurement Unit but rather the Attesting Environment end-point of the Trigger interface. See Section 4.1 for details on the Trigger interface.

* Measurement: Term introduced by RATS. In the context of this document, it can mean a representation of a physical property, the result of a test, the detection of an event, etc.

* Target Hardware Component: A hardware component which is a Target Environment for an Attesting Environment.

3. Use Cases

The solution presented in this document aims at mitigating two threats on hardware.

* Defective hardware components

Malfunctions of hardware components may be caused by environment and/or aging. Detection of such malfunctions is critical when relying on systems evolving in hazardous environments such as high pressure, extreme temperatures, contact with water, chemical substances or space radiations.

* Attacks on hardware components

Gaining control of the hardware of a system is particularly interesting for an attacker as it allows to tamper with the correct functioning of the system at a privileged level. Such control can be obtained by abusing software mechanisms or by having physical access to the system (particularly relevant for embedded systems) and using physical attack techniques.

Security and Safety note: Undetected hardware defects can compromise the integrity of cryptographic operations, attestation chains, or safety-critical controls, turning a physical fault into a security vulnerability or a life-threatening failure. In adversarial contexts, hardware degradation may also be leveraged to bypass attestation mechanisms or force a system into an exploitable state. Timely and verifiable detection of hardware component malfunctions is therefore critical for maintaining both operational safety and the trustworthiness of any attestation claim issued by a system.

For instance, environmental conditions and aging can alter the physical noise source of a TRNG, potentially reducing entropy and compromising the unpredictability required for security and safety. This TRNG example is extended in Section 7.2.

4. Attester Model

The RATS architecture presented in [RFC9334] introduces two types of environments in an Attester. The Attesting Environment (AE) and the Target Environment (TE). The Attesting Environment is in charge of collecting claims about a Target Environment. The Attesting Environment is then responsible for embedding those claims in an Evidence Conceptual Message.

This document focuses on claims used to represent the state of a target hardware component. Said claims can be related to physical properties (electromagnetic or thermal signature, timing values, power consumption, etc.), results of integrated self-tests or collected traces.

The goal of this section is to propose standard interfaces to trigger the computation of the measurement and to collect the computed measurement. Also, this section presents a mapping of Attesting Environments and Target Environments in different integration models of measurement mechanisms.

4.1. Abstract Representation

Mechanisms for collecting measurements of hardware components may highly depend of the type of hardware component and on the desired type of measurement. Therefore, this document proposes an abstract representation of such mechanisms. Considering a measurement mechanism as a black box with common interfaces, allows the content of this document to remain agnostic of the underlying mechanism and of the type of measurement collected while promoting interoperability with different real world implementations.

This document uses the following abstract objects:

* Measurement Unit

Black box used to represent a mechanism capable of computing measurements over a target. The Measurement Unit is part of the Attesting Environment.

* Trigger interface

To start the computation of a measurement, the Measurement Unit must be triggered. The trigger can follow an external request, a watchdog or any event set to trigger a measurement. The Measurement Unit receives a signal to start the computation of a measurement through the "Trigger" interface.

This interface is optional. For instance, it may not be used in case of continuous monitoring.

* Export interface

The "Export" interface allows a measurement to be exported from the Measurement Unit to a controlled memory region in the trust boundary of the Attesting Environment.

* Data Exchange channel

The measurement mechanism needs to have physical access on the property that it is in charge of measuring. The flow of data exchanged between the Measurement Unit and the target hardware component is represented by the "Data Exchange" channel. This channel is not accessible by the Attesting Environment excepted by the Measurement Unit.

4.2. Integration Models

The following subsections present possible layouts for integrating a Measurement Unit between the Attesting Environment and the target hardware component.

4.2.1. Embedded Measurement Unit

In this integration model, the Measurement Unit is part of the target hardware component. The separation between Measurement Unit (part of the Attesting Environment) and the Target Environment is only logical. The target hardware component and the Measurement Unit are part of the same die. Due to the proximity between the Measurement Unit and the target hardware component, the Data Exchange channel is not represented.

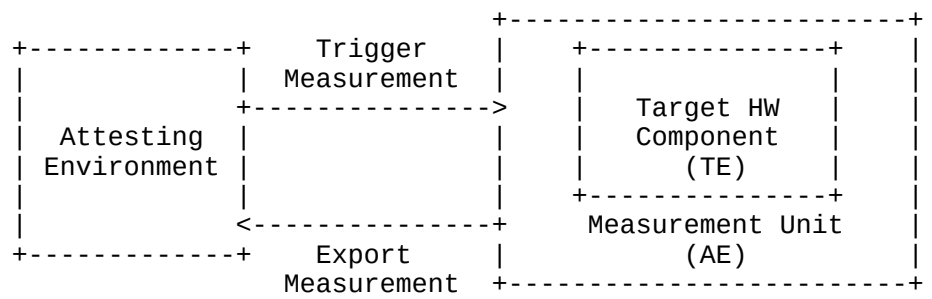

Figure 1: Abstract Representation of Embedded Measurement Unit

Ex: Different types of Built-In-Self-Tests (BIST) or Known Answer Tests (KAT).

Note: As shown in Figure 1, the Measurement Unit and target hardware component share the same die. Therefore, they are in the same physical security perimeter. This may have an impact on the trust model (see Section 8.4.3).

4.2.2. External Measurement Unit

In the following integration models, the Measurement Unit is external to the target hardware component.

4.2.2.1. Discrete Component

The Measurement Unit is a discrete component external to the target hardware component and to the Attesting Environment.

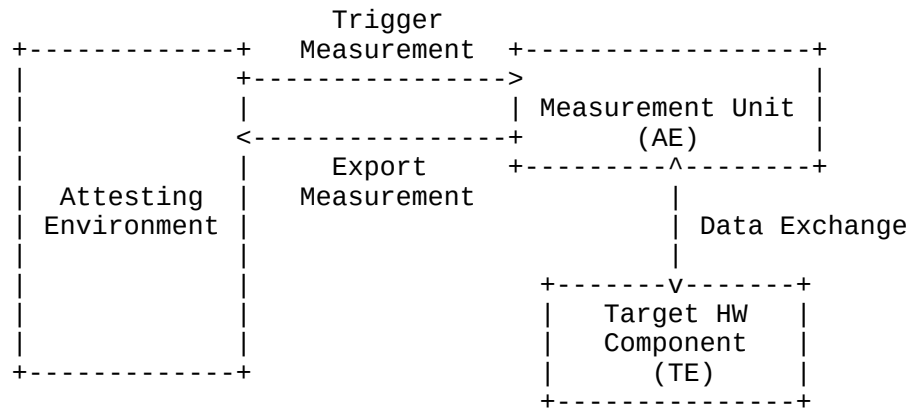

Figure 2: Abstract Representation of Discrete Measurement Unit

Ex: Sensors added on top of hardware component, Power Management IC (PMIC), Baseboard Management Controller (BMC).

In this integration model, the target hardware component and Measurement Unit can come from different sources (e.g., foundries). That can be leveraged to draw trust boundaries between the AE, TE and Measurement Unit. Using a discrete component implies the existence of physical communication channels between the AE, TE and Measurement Unit on which data such as measurement will transit. This introduces attack vectors. Refer to Section 8.

4.2.2.2. Integrated in Attesting Environment

The Measurement Unit is physically integrated in the Attesting Environment. It can take the form of hardware circuitry or be a software component. As the Measurement Unit is integrated in the Attesting Environment, the Trigger and Export interfaces are not represented in Figure 3.

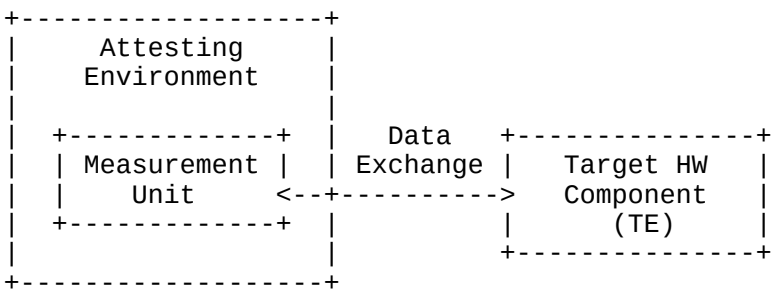

Figure 3: Abstract Representation of Measurement Unit Integrated in AE

Ex: Software logic (e.g., FIPS KAT)

Note: This integration model can be limiting in terms of what it is possible to measure.

Of course, both embedded and external Measurement Units can be found in the same system and possibly, a combination of embedded and external can be used to measure a single Target Environment (see the practical example in Section 7.2.4).

A single Attesting Environment can be responsible for one or more target hardware components. The Attesting Environment is therefore responsible for building Evidence for all of its target hardware components.

In addition to that, there may be multiple Attesting Environments. That case is discussed in [I-D.richardson-rats-composite-attesters].

4.3. Measurement Journey

Measurements of hardware components must be included in the Evidence to be sent to a Verifier. This implies that the Attesting Environment possesses a way to start the computation of the measurement (trigger), to securely retrieve the measurement (collection) and to securely embed the measurement in Evidence. During the completion of all these steps, the attacker has many opportunities to tamper with the integrity of the measurement or the execution logic (hardware or software).

Below are the identified steps of the journey of a measurement at the hardware level. These are important as this document implies a security model in which the attacker can tamper with hardware.

1. Trigger computation

   Measurement computation is triggered by an event (boot, external request, watchdog) or continuous. The Attesting Environment is able to trigger the computation of the measurement through the Trigger interface.

2. Compute measurement

   The measurement of the target hardware component is computed by the Measurement Unit.

3. Export measurement

   Once the measurement has been computed, it must be exported in order to be accessible by the Attesting Environment. The measurement transits from the Measurement Unit to the Attesting Environment through the export interface.

   Protection of the measurement in transit against tampering is critical for its trustworthiness. An attacker must not be able of tampering with the measurement in transit. This can be achieved by using bus protection techniques.

4. [optional] Store measurement

   It is possible that the measurement will not be directly included in Evidence but instead stored until it is effectively included in Evidence by the Attesting Environment.

   The measurement must be securely stored in the boundary of the Attesting Environment. An attacker must not be able of tampering with the measurement while it is at rest.

5. [optional] Process measurement

   It is possible that the measurement will not be directly included in Evidence as is but instead needs to be processed first before being effectively included in Evidence by the Attesting Environment.

   The processing of the measurement must be securely operated in the boundary of the Attesting Environment. An attacker must not be able of tampering with the processing logic.

6. Include measurement in Evidence

   The Attesting Environment is responsible for including the measurement data in Evidence. This operation must be carried out securely. An attacker must not be able to tamper with this logic.

   Note: At that point, the Evidence is not signed yet and could still be tampered by an attacker, possibly without being detected.

7. Sign Evidence

The signature operation must be carried out securely. An attacker must not be able to modify the content of the Evidence or forging signature for compromised data.

For instance, if the signature operation is offloaded to a remote hardware component and thus Evidence content must transit on a bus to reach this component, the attacker must not be able to manipulate data in transit (measurement, outputted signature).

Once stored in signed Evidence, the measurement is considered safe from unauthorized modification. This is because the cryptographic signature of the Evidence ensures integrity protection.

Figure 4 represents the steps of the measurement journey described above.

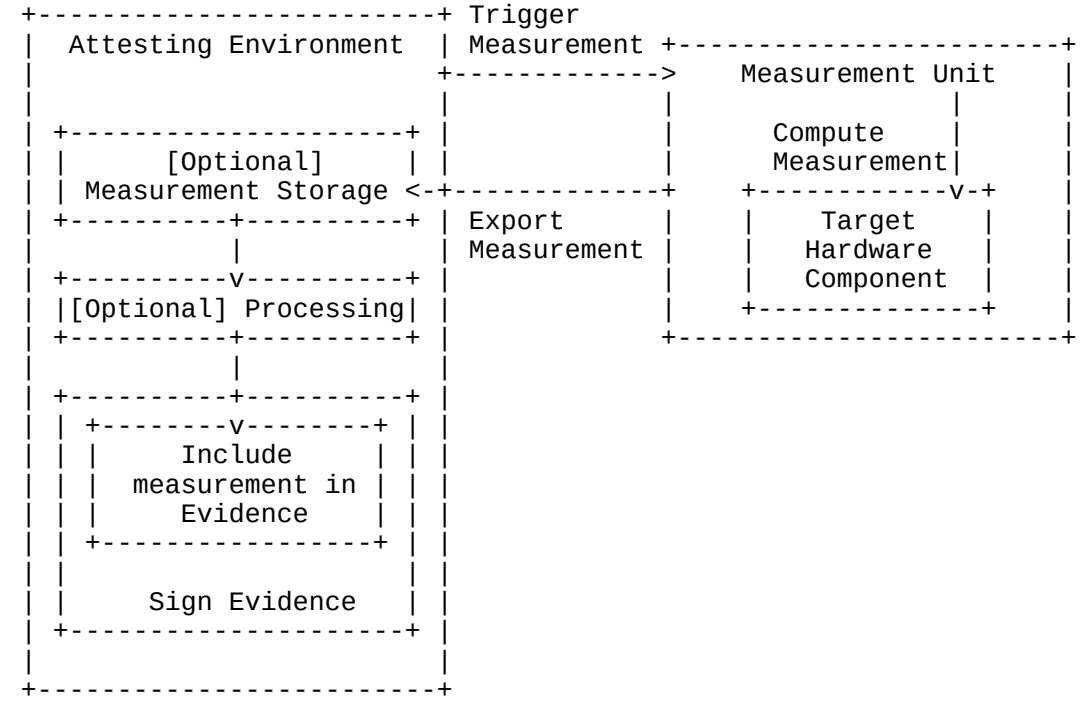

Figure 4: Measurement Journey

5. Inclusion in Conceptual Messages

This section introduces standard claims to be included in RATS Conceptual Messages. Conceptual Messages are defined in Section 8 of [RFC9334]. The RATS architecture does not mandate the usage of standard data formats for Conceptual Messages but protocols may require specific formats. Nonetheless, RATS proposed CoRIM for Endorsements and Reference Values and EAT for Evidence as standard data models. The CoRIM is defined in [I-D.ietf-rats-corim] and the EAT is defined in [RFC9711].

To promote interoperability, the following sections showcase how to use the CoRIM and EAT data models in the context of this document.

5.1. Endorsement

Endorsements in the scope of this document contain metadata describing the characteristics of Measurement Units and target hardware components that are necessary for the Verifier to correctly appraise Evidence.

These may include environmental robustness properties (e.g., operating temperature range, resistance to environmental conditions), Measurement Unit characteristics (e.g., accuracy, precision, uncertainty, sampling rate, sample count, method), calibration data (e.g., calibration coefficients and drift models over operational context), semantics of measurements (e.g., unit, scale, interpretation) and where applicable, the characteristics of reference or behavioral models to be used by the Verifier during appraisal.

TODO: if the unit of the measurement is specified in here, is it possible to reuse IANA numbers for Sensor Measurement Lists: https://www.iana.org/assignments/senml/senml.xhtml

5.1.1. Concise Reference Integrity Manifest (CoRIM)

Endorsements can be written inside a CoMID Endorsed Values triple of a CoRIM (see Section 5.1.6 of [I-D.ietf-rats-corim]). The Endorsed Values triple holds one or more measurement-map that are used to write the Endorsements.

5.2. Reference Value

Reference Values must be computed in a secure environment.

The Reference Value computed must correspond to the value that will be outputted in the expected environment of the system once in mission. For instance, a measurement might be dependent of the environmental conditions surrounding the system. This must be taken into account as a measurement different from the Reference Value does not necessarily mean bad behavior. If such context-dependent parameters cannot be foreseen, it is possible to include additional data in Evidence to give details about the context in which the measurement has been computed (see Section 5.3.1.4.1.2). The Verifier will then use these additional data to select the Reference Value that should be used in the context described by the additional data (kind of conditional Reference Values). This implies that attacker cannot modify these additional data otherwise, an attacker would be able to fool a Verifier into choosing Reference Values that don’t correspond to the actual context of the system.

Depending on the type of measurement and target hardware component, the Reference Value can be a value or a range or a function* of the operational context of the system and can correspond to a class, a group or an instance of target hardware component.

*could even be a ML model to detect abnormal physical properties depending on operational context of the system.

5.2.1. Concise Reference Integrity Manifest (CoRIM)

Reference Values can be written inside a CoMID Reference Values triple of a CoRIM (see Section 5.1.5 of [I-D.ietf-rats-corim]). The Reference Values triple holds one or more measurement-map that are used to write the Reference Values.

5.3. Evidence

The current version of this document proposes several approaches for including hardware component measurements in Evidence. For now, these options are present as brainstorming, to explore the different possibilities and may be removed in future versions of this document.

5.3.1. Entity Attestation Token (EAT)

5.3.1.1. Using EAT Measured Component Claim

It is possible for some measurements to be represented in an already existing EAT Measured Component. The EAT Measured Component is defined in [I-D.ietf-rats-eat-measured-component].

For instance, a custom measurement structure can be used to hold hardware component measurement in the "measurement" field of the "measured-component" structure from [I-D.ietf-rats-eat-measured-component]. Also, the flag field can be used to extend the measured-component base type with profile-defined semantics.

5.3.1.2. Using EAT Measurement Result Claim

It is possible for some measurements to be represented in an already existing EAT Measurement Result. The EAT Measurement Result is defined in Section 4.2.17 of [RFC9711].

This claim could be well-suited for measurements with on-device comparisons with reference values. For instance, self-tests (e.g., BIST, KAT) verify that the computed measurement corresponds to an expected value and output results such as "success" or "failure". In that case, a Measurement Result claim can be used.

5.3.1.3. Using EAT Submodule Claim

TODO it seems that the EAT Submodule can be used to embed hardware components claims (maybe only more complete subsystems not simple hardware components). Research if it could be extended to include measurements. Especially relevant if the submodule does not have its own Attesting Environment (Section 4.2.18 of [RFC9711]).

5.3.1.4. Using Measured Hardware Component Claims

This section proposes a new claim, the "Measured Hardware Component", to represent what is described in this document. This claim is presented in case the already existing claims mentioned above are not sufficient to correctly report measurements of hardware components.

The "measured hardware component" claim is inspired from the "measured component" claim introduced in [I-D.ietf-rats-eat-measured-component].

5.3.1.4.1. Information Model

This section presents the information model of a "measured hardware component".

The information elements (IEs) that constitute a "measured hardware component" are described in Table 1.

IE	Description	Requirement Level
Component Name	The name given to the target hardware component.	REQUIRED
Operational Context	Additional information on the operational context of the component.	OPTIONAL
Measurement List	List of measurements for the target hardware component. Each element of the list is composed of a Measurement Type and of a Measurement Value.	REQUIRED

Table 1: Measured Hardware Component Information Elements

5.3.1.4.1.1. Component Name

Component Name is used to identify the target hardware component.

5.3.1.4.1.2. Operational Context

Additional information on the operational context of the component. These can be used by the Verifier to appraise measurements.

By being placed at this level of the Measured Hardware Component claim, the operational context is shared by every measurement of the target hardware component. It is important that the operational context sampled corresponds to the actual operational context at the time of measurement computations (i.e., sampling of the operational context and computation of the measurements must be executed simultaneously (approximately). Otherwise, Time-Of-Check to Time-Of-Use (TOCTOU) attacks would be possible).

A lot of data can be included in Operational Context such as environmental (e.g., temperature, humidity, radiation), electrical (e.g., voltage, clock frequency, power state), workload (e.g., workload level, type), temporal (timestamp, uptime), system operation (degraded mode, thermal throttling) contexts.

TODO fill table. Operational Context is highly dependent on what is needed by Verifier which depends on what is measured and also what are the available sensors etc. so it will either contain a lot of optional fields or be profile-specific.

Field Name	Description	Requirement Level

Table 2: Fields of the Operational Context

Use case example: the measurement may be subject to variations depending on environmental context such as temperature (ideally specified in Endorsements, see Section 5.1). A measurement value might be acceptable when computed in a context of extreme cold but not if computed at room temperature. The Verifier must therefore be aware of the temperature surrounding the component to decide if the measurement corresponds to good behavior or not. The Verifier will therefore base its appraisal on the environmental context reported in Operational Context.

Note: Information in Operational Context is sensitive and must have the same level of protection as the measurements.

5.3.1.4.1.3. Measurement List

Field Name	Description	Requirement Level
Measurement Unit Identifier	Identifier for the Measurement Unit used to obtain the measurement	REQUIRED
Measurement Type	The type of the measurement.	REQUIRED
Measurement Value	The Value of the measurement. The content of this field depends on the Measurement Type.	REQUIRED

Table 3: Content of Elements of the Measurement List

* Measurement Unit Identifier:

Identifier for the Measurement Unit used to compute the measurement.

For instance there may be multiple sensors used to measure a single property of the target hardware component. In that case, the Measurement Unit Identifier allows to identify the Measurement Unit that was used.

* Measurement Type:

Specifier for the type of the measurement.

For example, the type can be used to specify if the measurement is the result of a self-test, the sampling of a physical property, a trace or an event (see Section 6.1.1).

* Measurement Value:

The structure that holds the actual measurement. The structure of the Measurement Value depends on the Measurement Type.

5.3.2. X.509 Certificate

Appendix C.3 of [RFC9711] describes methods to encode EAT claims in an X.509 certificate. These methods can be used for the claims presented in Section 5.3.1.

Ex: DICE uses X.509 certificates with a custom extension to carry Evidence [TCG-DICE]. TLS and DTLS extended with remote attestation also use X.509 certificates with an attestation extension [I-D.fossati-tls-attestation].

6. CDDL Definitions

This section presents CDDL definitions for the Measured Hardware Component claim to be included in EAT Measurement claim.

6.1. Measured Hardware Component Claim

```
   measured-hw-component = {
         component-id-label => component-id
         ? operational-ctx-label => operational-ctx
         measurement-list-label => [ + hw-measurement ]
      }

   operational-ctx = {
      ; structure for operational context
   }

   hw-measurement = {
         measurement-unit-id => tstr
         measurement-type-label => measurement-type   ; a choice
         measurement-value-label => measurement-value ; depends of type
      }

   measurement-type =
      mt-self-test /
      mt-phys-prop /
      mt-event /
      mt-trace /
      mt-other ; profile

   measurement-value = ; structure that depends of type
      mv-self-test /
      mv-phys-prop /
      mv-event /
      mv-trace /
      mv-other ; profile
```

Figure 5: CDDL of Measured Hardware Component Claim

6.1.1. Measurement Value

6.1.1.1. Physical Property

CDDL definition of the structure of measurement-value when measurement-type = mt-phys-prop.

```
   mv-phys-prop = {
      physical-property-id => tstr,
      value => number / bstr / tstr,
      ; unit, precision, scale, uncertainty,
      ; are already specified in Endorsements
   }
```

Figure 6: CDDL of Physical Property Measurement Value

6.1.1.2. Self-Test

CDDL definition of the structure of measurement-value when measurement-type = mt-self-test.

```
mv-self-test = {
   test-id => tstr,
   test-result-label => self-test-result,
}

self-test-result =
    st-pass /
    st-fail /
    st-degraded /
    st-not-run /
    st-unknown
```

Figure 7: CDDL of Self-Test Measurement Value

6.1.1.3. Event

CDDL definition of the structure of measurement-value when measurement-type = mt-event.

```
mv-event = {
  event-id => tstr,
  event-status-label => event-status,
  ? event-count => uint,
  ? event-time => int / uint
}

event-status =
    e-detected /
    e-not-detected /
    e-active /
    e-inactive /
    e-unknown
```

Figure 8: CDDL of Event Measurement Value

6.1.1.4. Trace

CDDL definition of the structure of measurement-value when measurement-type = mt-trace.

```
mv-trace = {
  trace-type-label => trace-type,
  trace-data => bstr / tstr
}

trace-type =
    digest /
    summary /
    counter
```

Figure 9: CDDL of Trace Measurement Value

6.1.1.5. Other

CDDL definition of the structure of measurement-value when measurement-type = mt-other.

TODO additional structures could be defined in a profile. Simply, the Measurement Value could be raw bytes that the Verifier would understand (by using a profile).

6.2. Inclusion in EAT Measurement Claim

The CDDL defined in Section 6.1 extends the $measurements-body-cbor and $measurements-body-json EAT sockets to add support for the measured-hw-component to the Measurements claim (Section 4.2.16 of [RFC9711]).

```
mhwc-cbor = bytes .cbor measured-hw-component
mhhwc-json = text .json measured-hw-component

; EAT CBOR (‘.feature "cbor"‘)
$measurements-body-cbor /= mhwc-cbor

; EAT JSON (‘.feature "json"‘)
$measurements-body-json /= mhwc-json
```

Figure 10: CDDL Extension of EAT Measurement Body

7. Practical Examples

This section is for informational purposes only.

Note: There are many interesting examples of hardware monitoring in [ISO5891] that are not covered here but fall in the scope of this document.

7.1. Monitoring Physical Properties

7.1.1. Using a Discrete Component Sensor

In this scenario, the Measurement Unit is implemented as a discrete external sensor, such as a temperature sensor or a Power Monitoring Integrated Circuit (PMIC). The Target Environment is the hardware component under observation, for example a CPU. This corresponds to the integration model described in Section 4.2.2.1.

The Measurement Unit observes physical properties of the Target Environment through a physical coupling, such as thermal conduction or electrical interaction (which in the model, corresponds to the Data Exchange channel), and exports digitized measurements to the Attesting Environment through the Export interface.

The Attesting Environment collects these measurements and includes them in Evidence, along with, if possible, contextual information describing the operational conditions under which the measurements were obtained.

In this model, the Measurement Unit is external to the Target Environment and may originate from a different manufacturer. As a result, the trustworthiness of the measurements depends on the integrity and characteristics of the sensor, which can be established through Endorsements describing its properties such as precision, calibration, and operating conditions (refer to Section 5.1).

During appraisal, the Verifier evaluates the measurements against Reference Values that may depend on the operational context. These Reference Values may be expressed as fixed ranges, condition-dependent functions, or behavioral models. The Verifier could use advanced models, including statistical or machine learning-based approaches, to detect anomalies (but such models would be part of the Appraisal Policy for Evidence and are not encoded in Evidence).

Note: compared to embedded Measurement Units, this model introduces additional attack surfaces, including sensor spoofing, manipulation of communication channels, and environmental interference. These risks must be considered in the system design and threat model (see Section 8).

7.1.2. Using an Embedded Sensor

7.1.2.1. Generic Example

In this scenario, the Measurement Unit is implemented as an on-die sensor integrated within the target hardware component. This corresponds to the embedded Measurement Unit model described Section 4.2.1.

The Measurement Unit observes physical properties of the Target Environment, such as temperature, voltage, or timing behavior, through direct internal coupling. Measurements are computed and made available to the Attesting Environment through internal interfaces, such as memory-mapped registers, without traversing external communication channels.

The Attesting Environment collects these measurements and includes them in Evidence, optionally along with operational context information to support appraisal.

As the Measurement Unit is physically integrated within the Target Environment, both share the same trust domain and physical security perimeter. The integrity of the Measurement Unit is typically established through Endorsements rather than through runtime measurement.

During appraisal, the Verifier evaluates the measurements against Reference Values that may depend on the operational context. As with other physical measurements, these Reference Values may be expressed as ranges, condition-dependent functions, or behavioral models.

Note: Compared to external sensors, this model reduces the attack surface by eliminating external communication channels and increasing the binding between the measurement and the component. However, it also reduces independence, as both the Target Environment and the Measurement Unit may be affected by the same faults or compromises. For instance, refer to Section 8.4.3.

7.1.2.2. Using a Ring Oscillator

In this scenario, the Measurement Unit is implemented as a ring oscillator integrated within the Target Environment. This corresponds to the embedded Measurement Unit model described Section 4.2.1. The oscillator frequency depends on physical and electrical properties of the hardware, including voltage, temperature, and process variations.

The Measurement Unit produces a digital representation of its oscillation frequency, which is collected by the Attesting Environment through an embodiment of the Export interface, then

included in Evidence. These measurements provide an indirect observation of the physical state of the component and can be used to detect anomalies such as voltage glitches, thermal variations, or aging effects.

During appraisal, the Verifier evaluates the reported frequency against Reference Values that depend on the operational context and calibration data provided through Endorsements.

Note: As this model is very sensitive to physical perturbations, deviations may have multiple possible causes. Therefore, the interpretation of measurements requires operational context. Ring oscillator measurements can then be used to complement other measurement types by providing continuous monitoring of the hardware physical and electrical behavior.

7.2. Detection by Self-Testing

This section provides practical examples that demonstrate how self-tests can be leveraged to measure a hardware component.

7.2.1. Using Built-In-Self-Tests (BIST)

In this scenario, the Measurement Unit is implemented as Built-In Self-Test (BIST) circuitry integrated within the Target Environment, for example within a memory subsystem or a cryptographic accelerator. This corresponding to the embedded Measurement Unit integration model described in Section 4.2.1.

The BIST logic executes predefined test patterns and compares the observed behavior of the component against expected results, producing a pass/fail outcome or a diagnostic signature. These tests may be executed at boot time or periodically during runtime.

The Attesting Environment collects the BIST results and includes them in Evidence as measurements associated with the corresponding target hardware component structured according to a Measurement Result or a Measured Hardware Component claim defined respectively in Section 5.3.1.2 and Section 5.3.1.4.

During appraisal, the Verifier compares the reported test results against Reference Values, typically expecting a successful outcome. Unlike measurements of physical properties, BIST results are deterministic and do not require contextual interpretation. In such case, the operational context is not used.

Note: This model provides strong assurance of functional correctness of the target hardware component and complements physical measurements by detecting faults that may not be observable through sensors.

7.2.2. TRNG Entropy Evaluation by an Embedded Measurement Unit

In this scenario, the Target Environment is the TRNG hardware component, whose entropy source constitutes the subject of the measurement. The Measurement Unit is implemented as on-die hardware logic tightly coupled to the TRNG, corresponding to the embedded Measurement Unit integration model described in Section 4.2.1.

The Measurement Unit continuously or periodically evaluates the entropy source by executing health tests and entropy estimators (Section 4 of [NIST-SP-800-90B]). Measurements are obtained through a Data Exchange channel, without traversing any software-accessible bus, and are exported directly to the Attesting Environment via the Export interface.

The Attesting Environment is the system ROM, which collects the measurement outputs and embeds them into Evidence (EAT, X.509 certificate). The Evidence includes multiple measurements for the TRNG component, such as health test results and minimum entropy values, structured according to the Measurement Result or Measured Hardware Component claim defined respectively in Section 5.3.1.2 and Section 5.3.1.4.

During appraisal, the Verifier validates the signature using the manufacturer's endorsement chain, then evaluates the measurements against Reference Values. Health test results are expected to indicate a passing state, and entropy values are compared against policy-defined thresholds.

Note: In this integration model, the Measurement Unit and Target Environment share the same physical security perimeter. As a result, the integrity of the Measurement Unit is not independently verified at runtime but is instead covered by manufacturer endorsements.

7.2.3. TRNG Entropy Evaluation by a Software Measurement Unit

In this scenario, the Target Environment is the TRNG, while the Measurement Unit is implemented as a software component executing within the Attesting Environment. This corresponds to the integration model described in Section 4.2.2.2.

The Measurement Unit obtains raw samples from the TRNG via a software interface and computes entropy-related measurements. As the Measurement Unit operates in software, its integrity must be established before its output can be trusted. This falls in the category of classical software measurements already specified by RATS documents.

The Evidence includes the entropy measurements produced by the software Measurement Unit.

During appraisal, the Verifier first evaluates the integrity of the Measurement Unit by comparing the reported digest against reference values obtained from a CoRIM. Only if the Measurement Unit is recognized as intact does the Verifier proceed to evaluate the entropy measurements.

This model introduces a dependency between the trustworthiness of the Measurement Unit and the validity of the measurements it produces. This dependency is always present but the trustworthiness of the MU is not always quantifiable (e.g., the MU cannot be measured), see Section 8.1.

7.2.4. TRNG Entropy Cross-Validation by Dual Measurement Units

This scenario is a mix of the two previous ones. The Target Environment is the TRNG, while two Measurement Units operate concurrently: a hardware Measurement Unit embedded in the component and a software Measurement Unit integrated within the Attesting Environment. This corresponds to a combination of the integration models described in Section 4.2.1 and Section 4.2.2.2.

Each Measurement Unit independently computes entropy-related measurements based on the same underlying noise source. The Attesting Environment collects both sets of measurements, associates them with their respective Measurement Unit identifiers, and aggregates them into a single Evidence structure.

This model enables cross-validation of measurements and allows the detection of silent failures affecting either one of the Measurement Units. A significant divergence between the two measurements may indicate faults, degradation, or inconsistencies in the measurement process, even when individual measurements satisfy their respective thresholds.

The Evidence, in this case, contains multiple measurements for the same target hardware component, originating from distinct Measurement Units.

7.3. Detection of Active Tampering

In this generic scenario, the Measurement Unit consists of tamper detection circuitry, such as an active mesh, voltage glitch detector, or light sensor, integrated within the hardware component.

These mechanisms do not produce measurements of physical properties but instead generate event-driven signals indicating potential tampering or fault conditions. Such signals may be triggered by physical intrusion, abnormal voltage or clock conditions, or environmental disturbances.

The Attesting Environment collects the status of these detectors and includes them in Evidence as security-relevant events or status indicators.

During appraisal, the Verifier interprets these signals according to an Appraisal Policy, typically treating any indication of tampering as a critical failure condition. Unlike other measurements, the absence of an alert does not guarantee the absence of an attack, but the presence of an alert provides strong evidence of compromise.

These mechanisms complement other measurement types by providing direct detection of active physical attacks and environmental anomalies.

7.4. Detection Using Traces

In this generic scenario, the Measurement Unit consists of hardware trace logic integrated within the Target Environment, such as Arm CoreSight, Intel Processor Trace, or Nexus trace modules.

These mechanisms observe the execution of the Target Environment and produce trace data reflecting instruction flow, memory accesses, or system events. Due to the high volume of trace data, the Attesting Environment typically processes or summarizes this information before including it in Evidence.

The resulting measurements may consist of aggregated statistics, cryptographic digests of trace segments, or derived indicators of anomalous behavior.

During appraisal, the Verifier evaluates these measurements against behavioral models describing expected execution patterns. These models may be expressed as statistical profiles or more advanced classifiers in the Appraisal Policy for Evidence.

Trace-based measurements provide insight into the runtime behavior of the Target Environment and can reveal anomalies that are not detectable through static measurements or physical sensors. However, they require careful processing and interpretation and may introduce additional considerations related to data volume, confidentiality, and trust in the trace collection infrastructure.

8. Security Considerations

The security considerations of RATS architecture apply (Section 12 of [RFC9334]). This section also mentions protection against physical attacks. These attacks are particularly relevant for this draft as collecting claims about hardware components implies a risk of physical compromise. Aging and action of environment on the system are also considered threats.

The security considerations of EAT Measured Component apply (Section 5 of [I-D.ietf-rats-eat-measured-component]) when using EAT Measured Component claim or Measured Hardware Component Claim.

The security considerations related to X.509 certificates apply (Section 8 of [RFC5280]) when using X.509 certificates to carry Evidence.

Security considerations of CoRIM apply (Section 11 of [I-D.ietf-rats-corim]) when using CORIM for Endorsements and Reference Values.

The following subsections are mainly focused on security considerations regarding the Attester during the steps of the Measurement Journey (see Section 4.3).

8.1. Root of Trust Components

Some components are essential for attestation (storage of attestation key, hardware Measurement Unit, etc.), if these are tampered with, there is no way to build trustworthy Evidence. These are considered the Root of Trust (RoT) for attestation because their correct functioning cannot be proved through attestation.

These are to be put in contrast with other components that are not critical for attestation (although they can be critical for the security of the system itself !).

8.2. Multiple Attesting Environments

In case of multiple Attesting Environments, distribution of freshness and binding of Evidence are discussed in [I-D.richardson-rats-composite-attesters].

8.3. Invasive Accesses

An attacker must not be able to leverage a Measurement Unit to access protected assets. For instance, access to protected assets can happen when computing measurements by using internal debug mechanisms (e.g., TAP controllers).

8.4. Threat Model

8.4.1. Software Attacks

There exist software attacks that can have a direct impact on hardware components' behavior. These are ideally mitigated by good and secure development practices but in case they happen, these attacks can be detected by monitoring physical properties of the component (such as power consumption, thermal and electromagnetic signatures, timing).

Ex: Software-induced Denial-of-Service (DoS).

Ex: Manipulation of privileged power control interface.

8.4.2. Physical Attacks

Physical attacks target the hardware of the system. They imply physical access to the system during its mission mode or while it is in the supply chain.

8.4.2.1. Passive Attacks

Passive physical attacks are used by attackers to leak information through analysis of system physical properties. Passive attacks, by definition, do not modify behavior of the system and therefore cannot alter the correct functioning of the attestation flow.

Ex: Side channel analysis of physical properties (EM emissions, power consumption, timing, temperature, probing, etc.)

The danger with passive attacks resides in the extraction of sensitive assets and particularly attestation key used to sign Evidence, which can be used for impersonation and Evidence forgery. This is already tackled in Section 12.1.1 of [RFC9334].

8.4.2.2. Active Attacks

Active physical attacks are the main problem since they allow an attacker (or a “natural” physical event) to tamper with the integrity of assets and execution flows of the system. These may therefore modify measurements in transit or at rest, inject arbitrary data in Evidence or bypass sensitive operations.

Ex: Attacks on bus (Active man-in-the-middle (MITM), injection, probing) or anywhere measurements are in transit before being integrated in a structure that cannot be tampered or spoofed (signed Evidence).

Ex: Glitching, fault injections to induce malicious behavior. May tamper with the target hardware component itself or the Measurement Unit or the logic used to build Evidence.

Ex: Memory tampering attacks to modify stored measurements.

Some techniques to mitigate physical attacks are usage of a Trusted Platform Module (TPM) or secure element for storage and correct execution of protected logic, bus protections, redundancy, sensors, active meshes, nose injection, etc. Note that some of these mitigations cannot directly prevent attacks but can be used for detection.

8.4.3. Supply Chain Attacks

Each stage of the supply chain introduces a new opportunity for an attacker to tamper with the produced system.

Supply chains attacks may lead to the injection of Trojans. Once a Trojan has been triggered, its activity may be reflected on the physical properties of the component (modified timing, different power consumption). It is therefore possible, in some cases, to detect an active Trojan by comparing the physical properties of the component when the Trojan is active against the reference physical properties of the component. Note that, if the Measurement Unit is part of the component itself, which means that it has been integrated by the foundry that introduced the Trojan, then it cannot be trusted.

9. Privacy Considerations

The privacy considerations of RATS architecture apply (Section 11 of [RFC9334]).

The privacy considerations of EAT Measured Component apply (Section 6 of [I-D.ietf-rats-eat-measured-component]) when using EAT Measured Component claim or Measured Hardware Component Claim.

Privacy considerations of CoRIM apply (Section 11 of [I-D.ietf-rats-corim]) when using CORIM for Endorsements and Reference Values.

TODO for reused claims privacy considerations are probably specified in other documents so refer to them.

TODO In new claims, some fields may be dangerous for privacy. Some fields may enable tracking.

10. Operational Considerations

It is possible that some measurement mechanisms may not be fully deterministic or may fail on rare occurrences or raise false positives.

It is also possible that aging or environmental context affect sensors.

These considerations must be taken into account and mitigated to an acceptable level by the designer.

11. IANA Considerations

This document has no IANA actions. TODO need IANA actions for claims defined in this document ?

12. References

12.1. Normative References

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

[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>.

[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>.

12.2. Informative References

[I-D.fossati-tls-attestation]
Tschofenig, H., Sheffer, Y., Howard, P., Mihalcea, I., Deshpande, Y., Niemi, A., and T. Fossati, "Using Attestation in Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)", Work in Progress, Internet-Draft, draft-fossati-tls-attestation-09, 30 April 2025, <https://datatracker.ietf.org/doc/html/draft-fossati-tls-attestation-09>.

[I-D.ietf-rats-corim]
Birkholz, H., Fossati, T., Deshpande, Y., Smith, N., and W. Pan, "Concise Reference Integrity Manifest", Work in Progress, Internet-Draft, draft-ietf-rats-corim-10, 2 March 2026, <https://datatracker.ietf.org/doc/html/draft-ietf-rats-corim-10>.

[I-D.ietf-rats-eat-measured-component]
Frost, S., Fossati, T., Tschofenig, H., and H. Birkholz, "Entity Attestation Token (EAT) Measured Component", Work in Progress, Internet-Draft, draft-ietf-rats-eat-measured-component-12, 20 February 2026, <https://datatracker.ietf.org/doc/html/draft-ietf-rats-eat-measured-component-12>.

[I-D.richardson-rats-composite-attesters]
Richardson, M., Birkholz, H., Deshpande, Y., and T. Fossati, "Taxonomy of Composite Attesters", Work in Progress, Internet-Draft, draft-richardson-rats-composite-attesters-04, 2 March 2026, <https://datatracker.ietf.org/doc/html/draft-richardson-rats-composite-attesters-04>.

[ISO5891] International Standards Organization, "ISO/IEC TR 5891:2024, Information security, cybersecurity and privacy protection — Hardware monitoring technology for hardware security assessment", April 2024, <https://www.iso.org/fr/standard/81806.html>.

[NIST-SP-800-90B]
National Institute of Standards and Technology, "Recommendation for the Entropy Sources Used for Random Bit Generation", January 2018, <https://nvlpubs.nist.gov/nistpubs/SpecialPublications/nist.sp.800-90b.pdf>.

[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/rfc/rfc5280>.

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

[TCG-DICE] Trusted Computing Group, "DICE Attestation Architecture, Version 1.00, Revision 0.23", March 2021, <https://trustedcomputinggroup.org/wp-content/uploads/DICE-Attestation-Architecture-r23-final.pdf>.

Appendix A. Collected CDDL

This appendix contains all the CDDL definitions included in this document.

```
mhwc-cbor = bytes .cbor measured-hw-component
mhhwc-json = text .json measured-hw-component

; EAT CBOR (‘.feature "cbor"‘)
$measurements-body-cbor /= mhwc-cbor

; EAT JSON (‘.feature "json"‘)
$measurements-body-json /= mhwc-json

measured-hw-component = {
      component-id-label => component-id
      ? operational-ctx-label => operational-ctx
      measurement-list-label => [ + hw-measurement ]
   }

operational-ctx = {
   ; structure for operational context
}

hw-measurement = {
      measurement-unit-id => tstr
      measurement-type-label => measurement-type   ; a choice
      measurement-value-label => measurement-value ; depends of type
   }

measurement-type =
   mt-self-test /
```

```
      mt-phys-prop /
      mt-event /
      mt-trace /
      mt-other ; profile

   measurement-value = ; structure that depends of type
      mv-self-test /
      mv-phys-prop /
      mv-event /
      mv-trace /
      mv-other ; profile

   mv-phys-prop = {
      physical-property-id => tstr,
      value => number / bstr / tstr,
      ; unit, precision, scale, uncertainty,
      ; are already specified in Endorsements
   }

   mv-self-test = {
      test-id => tstr,
      test-result-label => self-test-result,
   }

   self-test-result =
       st-pass /
       st-fail /
       st-degraded /
       st-not-run /
       st-unknown

   mv-event = {
     event-id => tstr,
     event-status-label => event-status,
     ? event-count => uint,
     ? event-time => int / uint
   }

   event-status =
       e-detected /
       e-not-detected /
       e-active /
       e-inactive /
       e-unknown

   mv-trace = {
     trace-type-label => trace-type,
     trace-data => bstr / tstr
```

```
   }

   trace-type =
       digest /
       summary /
       counter
```

Acknowledgments

Many thanks to Sylvain Guilley for reviewing the document and providing valuable comments.

Authors' Addresses

Antoine Poulain
Secure-IC
Email: antoine.poulain@secure-ic.com

Abdellah Kaci
Secure-IC
Email: abdellah.kaci@secure-ic.com