Stream:	Independent Submission				
RFC:	9783				
Category:	Informational				
Published:	May 2025				
ISSN:	2070-1721				
Authors:	H. Tschofenig	S. Frost	M. Brossard	A. Shaw	T. Fossati
	H-BRS	*Arm Limited*	*Arm Limited*	*HP Labs*	*Linaro*

RFC 9783
Arm's Platform Security Architecture (PSA) Attestation Token

Abstract

Arm's Platform Security Architecture (PSA) is a family of hardware and firmware security specifications, along with open-source reference implementations, aimed at helping device makers and chip manufacturers integrate best-practice security into their products. Devices that comply with PSA can generate attestation tokens as described in this document, which serve as the foundation for various protocols, including secure provisioning and network access control. This document specifies the structure and semantics of the PSA attestation token.

The PSA attestation token is a profile of the Entity Attestation Token (EAT). This specification describes the claims used in an attestation token generated by PSA-compliant systems, how these claims are serialized for transmission, and how they are cryptographically protected.

This Informational document is published as an Independent Submission to improve interoperability with Arm's architecture. It is not a standard nor a product of the IETF.

Status of This Memo

This document is not an Internet Standards Track specification; it is published for informational purposes.

This is a contribution to the RFC Series, independently of any other RFC stream. The RFC Editor has chosen to publish this document at its discretion and makes no statement about its value for implementation or deployment. Documents approved for publication by the RFC Editor are not candidates for any level of Internet Standard; see Section 2 of RFC 7841.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at https://www.rfc-editor.org/info/rfc9783.

Copyright Notice

Table of Contents

1. Introduction

The Platform Security Architecture (PSA) [PSA] is a set of hardware and firmware specifications backed by reference implementations and a security certification program [PSACertified]. The security specifications have been published by Arm, while the certification program and reference implementations are the result of a collaborative effort by companies from multiple sectors, including evaluation laboratories, IP semiconductor vendors, and security consultancies. The main objective of the PSA initiative is to assist device manufacturers and chip makers in incorporating best-practice security measures into their products.

Many devices now have Trusted Execution Environments (TEEs) that provide a safe space for security-sensitive code, such as cryptography, secure boot, secure storage, and other essential security functions. These security functions are typically exposed through a narrow and well-defined interface, and can be used by operating system libraries and applications.

As outlined in the Remote ATtestation procedureS (RATS) Architecture [RFC9334], an Attester produces a signed collection of Claims that constitutes Evidence about its Target Environment. This document focuses on the output provided by PSA's Initial Attestation API [PSA-API]. This output corresponds to Evidence in [RFC9334] and, as a design decision, the PSA attestation token is a profile of the Entity Attestation Token (EAT) [EAT]. Note that there are other profiles of EAT available for use with different use cases and by different attestation technologies, such as [RATS-TDX] and [RATS-QWESTOKEN].

Since the PSA tokens are also consumed by services outside the device, there is an actual need to ensure interoperability. Interoperability needs are addressed here by describing the exact syntax and semantics of the attestation claims, and defining the way these claims are encoded and cryptographically protected.

Further details on concepts expressed below can be found in the PSA Security Model documentation [PSA-SM].

As mentioned in the abstract, this memo documents a vendor extension to the RATS architecture and is not a standard.

2. Conventions and Definitions

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.

The terms Attester, Relying Party, Verifier, Attestation Result, Target Environment, Attesting Environment, and Evidence are defined in [RFC9334]. We use the term "receiver" to refer to Relying Parties and Verifiers.

We use the terms Evidence, "PSA attestation token", and "PSA token" interchangeably. The terms "sender" and Attester are used interchangeably. Likewise, we use the terms Verifier and "verification service" interchangeably.

Root of Trust (RoT):
: The minimal set of software, hardware, and data that has to be implicitly trusted in the platform; there is no software or hardware at a deeper level that can verify that the RoT is authentic and unmodified. An example of RoT is an initial bootloader in ROM, which contains cryptographic functions and credentials, running on a specific hardware platform.

Secure Processing Environment (SPE):
: A platform's processing environment for software that provides confidentiality and integrity for its runtime state, from software and hardware, outside of the SPE. Contains trusted code and trusted hardware. (Equivalent to a TEE, "secure world", or "secure enclave".)

Non-Secure Processing Environment (NSPE):
: The security domain (Application domain) outside of the SPE that typically contains the application firmware, real-time operating systems, applications, and general hardware. (Equivalent to Rich Execution Environment (REE), or "normal world".)

In this document, the structure of data is specified in Concise Data Definition Language (CDDL) [RFC8610].

3. PSA Attester Model

Figure 1 outlines the structure of the PSA Attester according to the conceptual model described in Section 3.1 of [RFC9334].

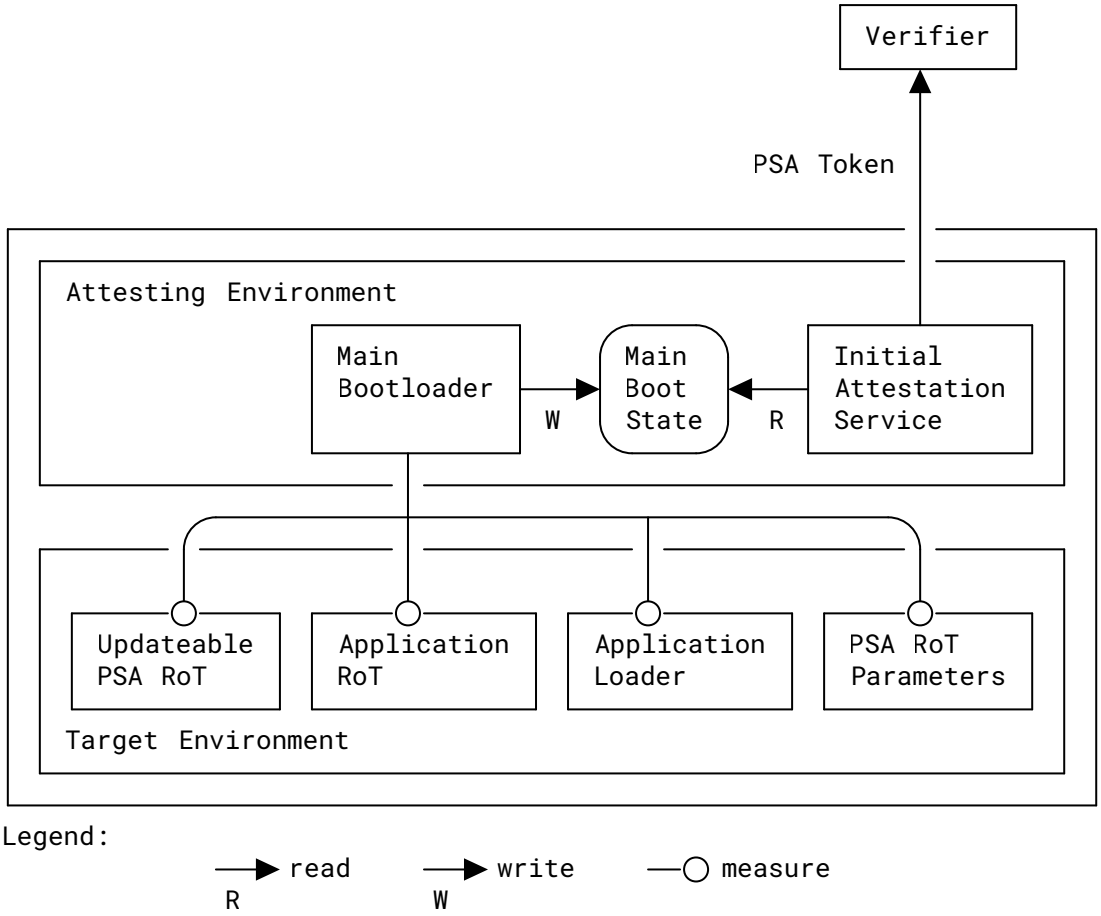

Figure 1: PSA Attester

The PSA Attester is a relatively straightforward embodiment of the RATS Attester with exactly one Attesting Environment and one or more Target Environments.

The Attesting Environment is responsible for collecting the information to be represented in PSA claims and to assemble them into Evidence. The Attesting Environment is made of two cooperating components:

- Executing at boot-time, the Main Bootloader measures the Target Environments (i.e., loaded software components and all the relevant PSA RoT parameters) and stores the recorded information in secure memory (Main Boot State). See Figure 2.

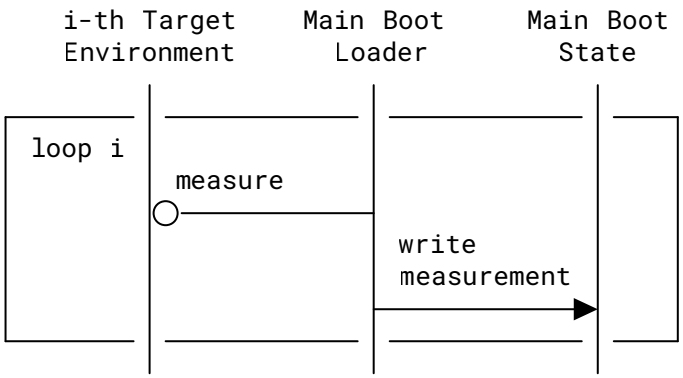

Figure 2: PSA Attester Boot Phase

- The Initial Attestation Service (executing at run-time in SPE) answers requests coming from NSPE via the PSA attestation API [PSA-API], collects and formats the claims from Main Boot State, and uses the Initial Attestation Key (IAK) to sign them and produce Evidence. See Figure 3.

The word "Initial" in "Initial Attestation Service" refers to a limited set of Target Environments, namely those representing the first foundational stages establishing the chain of trust of a PSA device. Collecting measurements from Target Environments after this initial phase is outside the scope of this specification. Extensions of this specification could collect up-to-date measurements from additional Target Environments and define additional claims for use within those environments, but these are, by definition, custom.

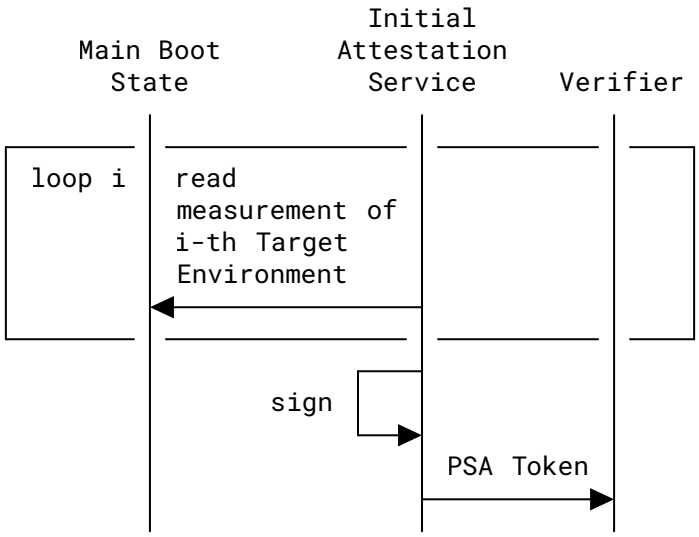

Figure 3: PSA Attester Run-Time Phase

The Target Environments can be of four types, some of which may or may not be present depending on the device architecture:

- (A subset of) the PSA RoT parameters, including Instance and Implementation IDs.
- The updateable PSA RoT, including the Secure Partition Manager and all PSA RoT services.
- The (optional) Application RoT, that is any application-defined security service possibly making use of the PSA RoT services.
- The loader of the application software running in NSPE.

A reference implementation of the PSA Attester is provided by [TF-M].

4. PSA Claims

This section describes the claims to be used in a PSA attestation token. A more comprehensive treatment of the EAT profiles defined by PSA is found in Section 5.

CDDL [RFC8610] along with text descriptions is used to define each claim independent of encoding. The following CDDL types are reused by different claims:

```
psa-hash-type = bytes .size 32 / bytes .size 48 / bytes .size 64
```

Two conventions are used to encode the Right-Hand-Side (RHS) of a claim. The postfix `-label` is used for EAT-defined claims and the postfix `-key` is used for PSA-originated claims.

4.1. Caller Claims

4.1.1. Nonce

The EAT [EAT] "nonce" (claim key 10) is used to carry the challenge provided by the caller to demonstrate freshness of the generated token.

Since the EAT nonce claim offers flexiblity for different attestation technologies, this specification applies the following constraints to the `nonce-type`:

- The length **MUST** be either 32, 48, or 64 bytes.
- Only a single nonce value is conveyed. The array notation **MUST NOT** be used for encoding the nonce value.

This claim **MUST** be present in a PSA attestation token.

```
psa-nonce = (
    nonce-label => psa-hash-type
)
```

4.1.2. Client ID

The Client ID claim represents the security domain of the caller.

In PSA, a security domain is represented by a signed integer whereby negative values represent callers from the NSPE and where positive IDs represent callers from the SPE. The value 0 is not permitted.

For an example definition of client IDs, see the PSA Firmware Framework [PSA-FF].

It is essential that this claim is checked in the verification process to ensure that a security domain, i.e., an attestation endpoint, cannot spoof a report from another security domain.

This claim **MUST** be present in a PSA attestation token.

```
psa-client-id-nspe-type = -2147483648...0
psa-client-id-spe-type = 1..2147483647

psa-client-id-type = psa-client-id-nspe-type / psa-client-id-spe-type

psa-client-id = (
    psa-client-id-key => psa-client-id-type
)
```

4.2. Target Identification Claims

4.2.1. Instance ID

The Instance ID claim represents the unique identifier of the IAK. The full definition is in [PSA-SM].

The EAT `ueid` (claim key 256) of type RAND is used. The following constraints apply to the `ueid-type`:

- The length **MUST** be 33 bytes.
- The first byte **MUST** be 0x01 (RAND) followed by the 32-byte unique identifier of the IAK. [PSA-API] provides implementation options for deriving the IAK unique identifier from the IAK itself.

This claim **MUST** be present in a PSA attestation token.

```
psa-instance-id-type = bytes .size 33

psa-instance-id = (
    ueid-label => psa-instance-id-type
)
```

4.2.2. Implementation ID

The Implementation ID claim uniquely identifies the hardware assembly of the immutable PSA RoT. A verification service uses this claim to locate the details of the PSA RoT implementation from an Endorser or manufacturer. Such details are used by a verification service to determine the security properties or certification status of the PSA RoT implementation.

The value and format of the ID is decided by the manufacturer or a particular certification scheme. For example, the ID could take the form of a product serial number, database ID, or other appropriate identifier.

This claim **MUST** be present in a PSA attestation token.

Note that this identifies the PSA RoT implementation, not a particular instance. To uniquely identify an instance, see the Instance ID claim Section 4.2.1.

```
psa-implementation-id-type = bytes .size 32

psa-implementation-id = (
    psa-implementation-id-key => psa-implementation-id-type
)
```

4.2.3. Certification Reference

The Certification Reference claim is used to link the class of chip and PSA RoT of the attesting device to an associated entry in the PSA Certification database. The Certification Reference claim **MUST** be represented as a string made of nineteen numeric characters: a thirteen-digit EAN-13 [EAN-13] followed by a dash "-" and the five-digit versioning information described in [PSA-Cert-Guide].

Linking to the PSA Certification entry can still be achieved if this claim is not present in the token by making an association at a Verifier between the reference value and other token claim values, for example, the Implementation ID.

This claim **MAY** be present in a PSA attestation token.

```
psa-certification-reference-type = text .regexp "[0-9]{13}-[0-9]{5}"

psa-certification-reference = (
    ? psa-certification-reference-key =>
        psa-certification-reference-type
)
```

4.3. Target State Claims

4.3.1. Security Lifecycle

The Security Lifecycle claim represents the current lifecycle state of the PSA RoT. The state is represented by an integer that is divided to convey a major state and a minor state. A major state is mandatory and defined by [PSA-SM]. A minor state is optional and 'IMPLEMENTATION DEFINED'. The PSA security lifecycle state and implementation state are encoded as follows:

- major[15:8] - PSA security lifecycle state, and
- minor[7:0] - IMPLEMENTATION DEFINED state.

The PSA lifecycle states are illustrated in Figure 4. For PSA, a Verifier can only trust reports from the PSA RoT when it is in SECURED or NON_PSA_ROT_DEBUG major states.

This claim **MUST** be present in a PSA attestation token.

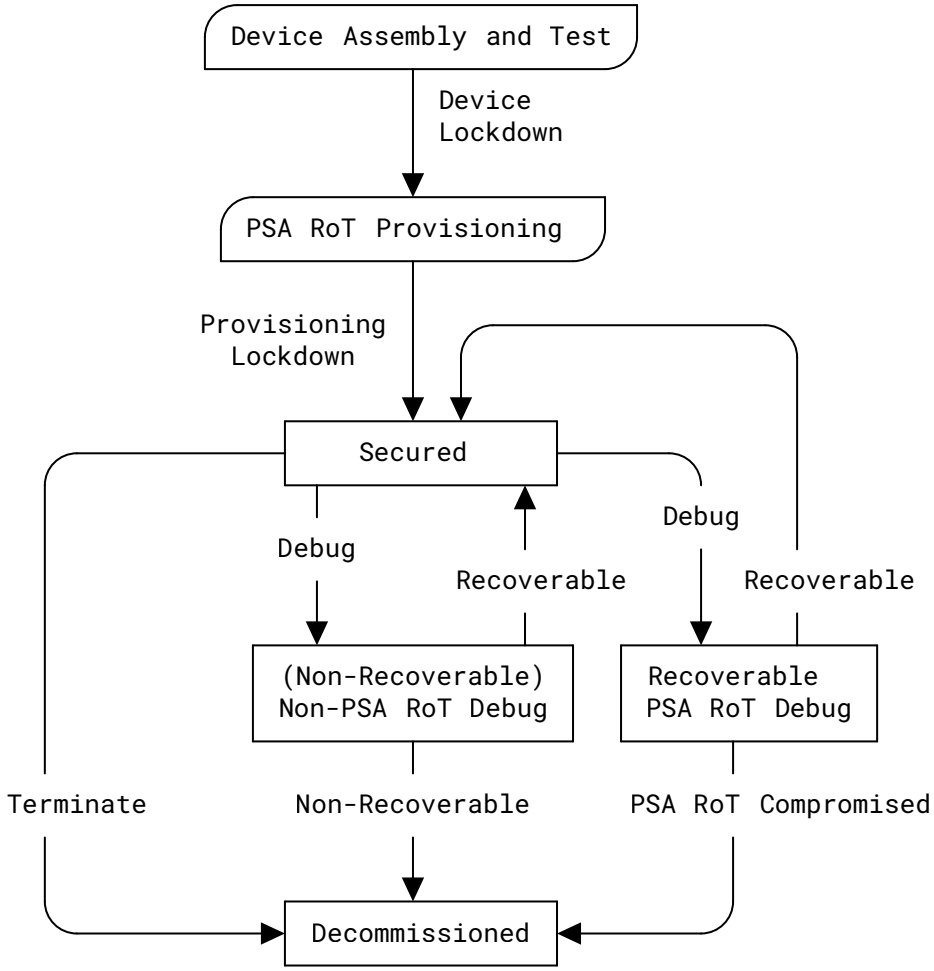

Figure 4: PSA Lifecycle States

The CDDL representation is shown below. Table 1 provides the mappings between Figure 4 and the data model.

```
psa-lifecycle-unknown-type = 0x0000..0x00ff
psa-lifecycle-assembly-and-test-type = 0x1000..0x10ff
psa-lifecycle-psa-rot-provisioning-type = 0x2000..0x20ff
psa-lifecycle-secured-type = 0x3000..0x30ff
psa-lifecycle-non-psa-rot-debug-type = 0x4000..0x40ff
psa-lifecycle-recoverable-psa-rot-debug-type = 0x5000..0x50ff
psa-lifecycle-decommissioned-type = 0x6000..0x60ff

psa-lifecycle-type =
    psa-lifecycle-unknown-type /
    psa-lifecycle-assembly-and-test-type /
    psa-lifecycle-psa-rot-provisioning-type /
    psa-lifecycle-secured-type /
    psa-lifecycle-non-psa-rot-debug-type /
    psa-lifecycle-recoverable-psa-rot-debug-type /
    psa-lifecycle-decommissioned-type

psa-lifecycle = (
    psa-lifecycle-key => psa-lifecycle-type
)
```

`psa-lifecycle-unknown-type` is not shown in Figure 4; it represents an invalid state that must not occur in a system.

CDDL	Lifecycle States
`psa-lifecycle-unknown-type`	
`psa-lifecycle-assembly-and-test-type`	Assembly and Test
`psa-lifecycle-psa-rot-provisioning-type`	PSA RoT Provisioning
`psa-lifecycle-secured-type`	Secured
`psa-lifecycle-non-psa-rot-debug-type`	Non-Recoverable PSA RoT Debug
`psa-lifecycle-recoverable-psa-rot-debug-type`	Recoverable PSA RoT Debug
`psa-lifecycle-decommissioned-type`	Decommissioned

Table 1: Lifecycle States Mappings

4.3.2. Boot Seed

The "bootseed" claim contains a value created at system boot time that allows differentiation of attestation reports from different boot sessions of a particular entity (i.e., a certain Instance ID).

The EAT "bootseed" (claim key 268) is used. The following constraints apply to the `binary-data` type:

- The length **MUST** be between 8 and 32 bytes.

This claim **MAY** be present in a PSA attestation token.

```
psa-boot-seed-type = bytes .size (8..32)

psa-boot-seed = (
    boot-seed-label => psa-boot-seed-type
)
```

4.4. Software Inventory Claims

4.4.1. Software Components

The Software Components claim is a list of software components that includes all the software (both code and configuration) loaded by the PSA RoT. This claim **MUST** be included in attestation tokens produced by an implementation conformant with [PSA-SM].

Each entry in the Software Components list describes one software component using the attributes described in the following subsections. Unless explicitly stated, the presence of an attribute is **OPTIONAL**.

Note that a Relying Party will typically see the result of the appraisal process from the Verifier in form of an Attestation Result rather than the PSA token from the attesting endpoint as described in [RFC9334]. Therefore, a Relying Party is not expected to understand the Software Components claim. Instead, it is for the Verifier to check this claim against the available Reference Values and provide an answer in form of a "high-level" Attestation Result, which may or may not include the original Software Components claim.

```
psa-software-component = {
  ? &(measurement-type: 1) => text
    &(measurement-value: 2) => psa-hash-type
  ? &(version: 4) => text
    &(signer-id: 5) => psa-hash-type
  ? &(measurement-desc: 6) => text
}

psa-software-components = (
    psa-software-components-key => [ + psa-software-component ]
)
```

4.4.1.1. Measurement Type

The Measurement Type attribute (key=1) is a short string representing the role of this software component.

The following measurement types **MAY** be used for code measurements:

"BL": a Boot Loader

"PRoT": a component of the PSA Root of Trust

"ARoT": a component of the Application Root of Trust

"App": a component of the NSPE application

"TS": a component of a Trusted Subsystem

The same labels with a "_CONFIG" postfix (e.g., "PRoT_CONFIG") **MAY** be used for configuration measurements.

This attribute **SHOULD** be present in a PSA software component unless there is a very good reason to leave it out, for example, in networks with severely constrained bandwidth where sparing a few bytes really makes a difference.

4.4.1.2. Measurement Value

The Measurement Value attribute (key=2) represents a hash of the invariant software component in memory at startup time. The value **MUST** be a cryptographic hash of 256 bits or stronger.

This attribute **MUST** be present in a PSA software component.

4.4.1.3. Version

The Version attribute (key=4) is the issued software version in the form of a text string. The value of this attribute will correspond to the entry in the original signed manifest of the component.

4.4.1.4. Signer ID

The Signer ID attribute (key=5) uniquely identifies the signer of the software component. The identification is typically accomplished by hashing the signer's public key. The value of this attribute will correspond to the entry in the original manifest for the component. This can be used by a Verifier to ensure the components were signed by an expected trusted source.

This attribute **MUST** be present in a PSA software component to be compliant with [PSA-SM].

4.4.1.5. Measurement Description

The Measurement Description attribute (key=6) contains a string identifying the hash algorithm used to compute the corresponding Measurement Value. The string **SHOULD** be encoded according to "Hash Name String" in the "Named Information Hash Algorithm Registry" [NAMED-INFO].

4.5. Verification Claims

The following claims, although part of Evidence, do not reflect the internal state of the Attester. Instead, they aim to help receivers, including Relying Parties, in processing the received attestation Evidence.

4.5.1. Verification Service Indicator

The Verification Service Indicator claim is a hint used by a Relying Party to locate a verification service for the token. The value is a text string that can be used to locate the service (typically, a URL specifying the address of the verification service API). A Relying Party may choose to ignore this claim in favor of other information.

```
psa-verification-service-indicator-type = text

psa-verification-service-indicator = (
    ? psa-verification-service-indicator-key =>
        psa-verification-service-indicator-type
)
```

It is assumed that the Relying Party is pre-configured with a list of trusted verification services and that the contents of this hint can be used to look up the correct one. Under no circumstances must the Relying Party be tricked into contacting an unknown and untrusted verification service since the returned Attestation Result cannot be relied on.

Note: This hint requires the Relying Party to parse the content of the PSA token. Since the Relying Party may not be in possession of a trust anchor to verify the digital signature, it uses the hint in the same way as it would treat any other information provided by an external party, which includes attacker-provided data.

4.5.2. Profile Definition

The Profile Definition claim encodes the unique identifier that corresponds to the EAT profile described by this document. This allows a receiver to assign the intended semantics to the rest of the claims found in the token.

The EAT `eat_profile` (claim key 265) is used.

The URI encoding **MUST** be used.

The value **MUST** be `tag:psacertified.org,2023:psa#tfm` for the profile defined in Section 5.2.

Future profiles derived from the baseline PSA profile **SHALL** create their unique value as described in Section 4.5.2.1.

This claim **MUST** be present in a PSA attestation token.

See Section 4.6 for considerations about backwards compatibility with previous versions of the PSA attestation token format.

```
psa-profile-type = "tag:psacertified.org,2023:psa#tfm"

psa-profile = (
    profile-label => psa-profile-type
)
```

4.5.2.1. URI Structure for the Derived Profile Identifiers

A new profile is associated with a unique string.

The string **MUST** use the URI fragment syntax defined in Section 3.5 of [RFC3986].

The string **SHOULD** be short to avoid unnecessary overhead.

To avoid collisions, profile authors **SHOULD** communicate their intent upfront to use a certain string that uses the inquiry form on the website [PSACertified].

To derive the value to be used for the `eat_profile` claim, the string is added as a fragment to the `tag:psacertified.org,2023:psa` tag URI [RFC4151].

For example, a hypothetical profile using only COSE_Mac0 with the AES Message Authentication Code (AES-MAC) may decide to use the string "aes-mac". The `eat_profile` value would then be `tag:psacertified.org,2023:psa#aes-mac`.

4.6. Backwards Compatibility Considerations

An earlier draft of this document [PSA-OLD] identified by the `PSA_IOT_PROFILE_1` profile, used claim key values from the "private use range" of the CWT Claims registry. These claim keys have now been deprecated.

Table 2 provides the mappings between the deprecated and new claim keys.

	PSA_IOT_PROFILE_1	tag:psacertified.org,2023:psa#tfm
Nonce	-75008	10 (EAT nonce)
Instance ID	-75009	256 (EAT euid)
Profile Definition	-75000	265 (EAT eat_profile)
Client ID	-75001	2394
Security Lifecycle	-75002	2395
Implementation ID	-75003	2396
Boot Seed	-75004	268 (EAT bootseed)
Certification Reference	-75005	2398

	PSA_IOT_PROFILE_1	tag:psacertified.org,2023:psa#tfm
Software Components	-75006	2399
Verification Service Indicator	-75010	2400

Table 2: Claim Key Mappings

The new profile introduces three further changes:

- The "bootseed" claim is now optional and of variable length (see Section 4.3.2).
- The "No Software Measurements" claim has been retired.
- The "Certification Reference" claim syntax changed from EAN-13 to EAN-13+5 (see Section 4.2.3).

To simplify the transition to the token format described in this document, it is **RECOMMENDED** that Verifiers accept tokens encoded according to the old profile (`PSA_IOT_PROFILE_1`) as well as to the profile defined in this document (`tag:psacertified.org,2023:psa#tfm`), at least for the time needed to their devices to upgrade.

5. Profiles

This document defines a baseline with common requirements that all PSA profiles must satisfy. (Note that this does not apply to [PSA-OLD].)

This document also defines a "TFM" profile (Section 5.2) that builds on the baseline while constraining the use of COSE algorithms to improve interoperability between Attesters and Verifiers.

Baseline and TFM are what the EAT calls a "partial" and "full" profile, respectively. See Section 6.2 of [EAT] for further details regarding profiles.

5.1. Baseline Profile

5.1.1. Token Encoding and Signing

The PSA attestation token is encoded in CBOR [STD94] format. The CBOR representation of a PSA token **MUST** be "valid" according to the definition in Section 1.2 of RFC 8949 [STD94]. Besides, only definite-length string, arrays, and maps are allowed. Given that a PSA Attester is typically found in a constrained device, it **MAY** NOT emit CBOR preferred serializations (Section 4.1 of RFC 8949 [STD94]). Therefore, the Verifier **MUST** be a variation-tolerant CBOR decoder.

Cryptographic protection is obtained by wrapping the `psa-token` claims set in a COSE Web Token (CWT) [RFC8392]. For asymmetric key algorithms, the signature structure **MUST** be a tagged (18) COSE_Sign1. For symmetric key algorithms, the signature structure **MUST** be a tagged (17) COSE_Mac0.

Acknowledging the variety of markets, regulations, and use cases in which the PSA attestation token can be used, the baseline profile does not impose any strong requirement on the cryptographic algorithms that need to be supported by Attesters and Verifiers. The flexibility provided by the COSE format should be sufficient to deal with the level of cryptographic agility needed to adapt to specific use cases. It is **RECOMMENDED** that commonly adopted algorithms are used, such as those discussed in [COSE-ALGS]. It is expected that receivers will accept a wider range of algorithms while Attesters would produce PSA tokens using only one such algorithm.

The CWT CBOR tag (61) is not used. An application that needs to exchange PSA attestation tokens can wrap the serialized COSE_Sign1 or COSE_Mac0 in the media type defined in Section 10.2 or the CoAP Content-Format defined in Section 10.3.

A PSA token is always directly signed by the PSA RoT. Therefore, a PSA-token claims set (Section 4) is never carried in a Detached EAT bundle (Section 5 of [EAT]).

5.1.2. Freshness Model

The PSA token supports the freshness models for attestation Evidence based on nonces and epoch handles (Section 10.2 and Section 10.3 of [RFC9334]) using the "nonce" claim to convey the nonce or epoch handle supplied by the Verifier. No further assumption on the specific remote attestation protocol is made.

Note that use of epoch handles is constrained by the type restrictions imposed by the `eat_nonce` syntax. For use in PSA tokens, it must be possible to encode the epoch handle as an opaque binary string between 8 and 64 octets.

5.1.3. Synopsis

Table 3 presents a concise view of the requirements described in the preceding sections.

Issue	Profile Definition
CBOR/JSON	CBOR **MUST** be used.
CBOR Encoding	Definite length maps and arrays **MUST** be used.
CBOR Encoding	Definite length strings **MUST** be used.
CBOR Serialization	Variant serialization **MAY** be used.
COSE Protection	COSE_Sign1 and/or COSE_Mac0 **MUST** be used.
Algorithms	[COSE-ALGS] **SHOULD** be used.
Detached EAT Bundle Usage	Detached EAT bundles **MUST NOT** be sent.

Issue	Profile Definition
Verification Key Identification	Any identification method listed in Appendix F.1 of [EAT].
Endorsements	See Section 8.2.
Freshness	Nonce or epoch ID-based.
Claims	Those defined in Section 4. As per general EAT rules, the receiver **MUST NOT** error out on claims it does not understand.

Table 3: Baseline Profile

5.2. Profile TFM

The TFM profile is appropriate for the code base implemented in [TF-M] and should apply for most derivative implementations. If an implementation changes the requirements described below, then a different profile value should be used (Section 4.5.2.1) to ensure interoperability. This includes a restriction of the profile to a subset of the COSE Protection scheme requirements.

Table 4 presents a concise view of the requirements.

The value of the `eat_profile` **MUST** be `tag:psacertified.org,2023:psa#tfm`.

Issue	Profile Definition
CBOR/JSON	See Section 5.1.
CBOR Encoding	See Section 5.1.
CBOR Encoding	See Section 5.1.
CBOR Serialization	See Section 5.1.
COSE Protection	COSE_Sign1 or COSE_Mac0 **MUST** be used.
Algorithms	The receiver **MUST** accept ES256, ES384, and ES512 with COSE_Sign1 and HMAC256/256, HMAC384/384, and HMAC512/512 with COSE_Mac0; the sender **MUST** send one of these.
Detached EAT Bundle Usage	See Section 5.1.
Verification Key Identification	Claim-Based Key Identification (Appendix F.1.4 of [EAT]) using Instance ID.
Endorsements	See Section 8.2.

Issue	Profile Definition
Freshness	See Section 5.1.
Claims	See Section 5.1.

Table 4: TF-M Profile

6. Collated CDDL

```
psa-token = {
    psa-nonce
    psa-instance-id
    psa-verification-service-indicator
    psa-profile
    psa-implementation-id
    psa-client-id
    psa-lifecycle
    psa-certification-reference
    ? psa-boot-seed
    psa-software-components
}

psa-client-id-key = 2394
psa-lifecycle-key = 2395
psa-implementation-id-key = 2396
psa-certification-reference-key = 2398
psa-software-components-key = 2399
psa-verification-service-indicator-key = 2400

nonce-label = 10
ueid-label = 256
boot-seed-label = 268
profile-label = 265

psa-hash-type = bytes .size 32 / bytes .size 48 / bytes .size 64

psa-boot-seed-type = bytes .size (8..32)

psa-boot-seed = (
    boot-seed-label => psa-boot-seed-type
)

psa-client-id-nspe-type = -2147483648...0
psa-client-id-spe-type = 1..2147483647

psa-client-id-type = psa-client-id-nspe-type / psa-client-id-spe-type

psa-client-id = (
    psa-client-id-key => psa-client-id-type
)

psa-certification-reference-type = text .regexp "[0-9]{13}-[0-9]{5}"

psa-certification-reference = (
```

```
    ? psa-certification-reference-key =>
        psa-certification-reference-type
)

psa-implementation-id-type = bytes .size 32

psa-implementation-id = (
    psa-implementation-id-key => psa-implementation-id-type
)

psa-instance-id-type = bytes .size 33

psa-instance-id = (
    ueid-label => psa-instance-id-type
)

psa-nonce = (
    nonce-label => psa-hash-type
)

psa-profile-type = "tag:psacertified.org,2023:psa#tfm"

psa-profile = (
    profile-label => psa-profile-type
)

psa-lifecycle-unknown-type = 0x0000..0x00ff
psa-lifecycle-assembly-and-test-type = 0x1000..0x10ff
psa-lifecycle-psa-rot-provisioning-type = 0x2000..0x20ff
psa-lifecycle-secured-type = 0x3000..0x30ff
psa-lifecycle-non-psa-rot-debug-type = 0x4000..0x40ff
psa-lifecycle-recoverable-psa-rot-debug-type = 0x5000..0x50ff
psa-lifecycle-decommissioned-type = 0x6000..0x60ff

psa-lifecycle-type =
    psa-lifecycle-unknown-type /
    psa-lifecycle-assembly-and-test-type /
    psa-lifecycle-psa-rot-provisioning-type /
    psa-lifecycle-secured-type /
    psa-lifecycle-non-psa-rot-debug-type /
    psa-lifecycle-recoverable-psa-rot-debug-type /
    psa-lifecycle-decommissioned-type

psa-lifecycle = (
    psa-lifecycle-key => psa-lifecycle-type
)

psa-software-component = {
  ? &(measurement-type: 1) => text
    &(measurement-value: 2) => psa-hash-type
  ? &(version: 4) => text
    &(signer-id: 5) => psa-hash-type
  ? &(measurement-desc: 6) => text
}

psa-software-components = (
    psa-software-components-key => [ + psa-software-component ]
)
```

```
psa-verification-service-indicator-type = text

psa-verification-service-indicator = (
    ? psa-verification-service-indicator-key =>
        psa-verification-service-indicator-type
)
```

7. Scalability Considerations

IAKs (see Section 3) can be either raw public keys or certified public keys.

Certified public keys require the manufacturer to run the certification authority (CA) that issues X.509 certificates for the IAKs. (Note that operating a CA is a complex and expensive task that may be unaffordable to certain manufacturers.)

Using certified public keys offers better scalability properties when compared to using raw public keys, namely:

- Storage requirements for the Verifier are minimized; the same manufacturer's trust anchor is used for any number of devices.
- The provisioning model is simpler and more robust since there is no need to notify the Verifier about each newly manufactured device.

Furthermore, existing and well-understood revocation mechanisms can be readily used.

The IAK's X.509 certificates can be inlined in the PSA token using the `x5chain` COSE header parameter [COSE-X509] at the cost of an increase in the PSA token size. Section 4.4 of [TLS12-IoT] and Section 15 of [TLS13-IoT] provide guidance for profiling X.509 certificates used in IoT deployments. Note that the exact split between pre-provisioned and inlined certificates may vary depending on the specific deployment. In that respect, `x5chain` is quite flexible. It can contain the end entity (EE) certification only, the EE and a partial chain, or the EE and the full chain up to the trust anchor (see Section 2 of [COSE-X509] for the details). Constraints around network bandwidth and computing resources available to endpoints, such as network buffers, may dictate a reasonable split point.

8. PSA Token Verification

To verify the token, the primary need is to check correct encoding and signing as detailed in Section 5.1.1. The key used for verification is either supplied to the Verifier by an authorized Endorser along with the corresponding Attester's Instance ID or inlined in the token using the `x5chain` header parameter as described in Section 7. If the IAK is a raw public key and the Instance ID claim is used to assist in locating the key used to verify the signature covering the CWT token. If the IAK is a certified public key, X.509 path construction and validation (Section 6 of [X509]) up to a trusted CA **MUST** be successful before the key is used to verify the token signature. This also includes revocation checking.

The Verifier typically has a policy where it compares some claims in this profile to reference values registered with it for a given deployment. This verification process serves to confirm that the device is endorsed by the manufacturer supply chain. The policy may require that the relevant claims must have a match to a registered reference value. All claims may be worthy of additional appraisal. It is likely that most deployments would include a policy with appraisal for the following claims:

- Implementation ID: The value of the Implementation ID can be used to identify the verification requirements of the deployment.
- Software Component, Measurement Value: This value can uniquely identify a firmware release from the supply chain. In some cases, a Verifier may maintain a record for a series of firmware releases being patches to an original baseline release. A verification policy may then allow this value to match any point on that release sequence or expect some minimum level of maturity related to the sequence.
- Software Component, Signer ID: Where present in a deployment, this could allow a Verifier to operate a more general policy than that for Measurement Value as above by allowing a token to contain any firmware entries signed by a known Signer ID without checking for a uniquely registered version.
- Certification Reference: If present, this value could be used as a hint to locate security certification information associated with the attesting device. An example could be a reference to a [PSACertified] certificate.

8.1. AR4SI Trustworthiness Claims Mappings

[RATS-AR4SI] defines an information model that Verifiers can employ to produce Attestation Results. AR4SI provides a set of standardized appraisal categories and tiers that greatly simplifies the task of writing Relying Party policies in Multi-Attester environments.

The contents of Table 5 are intended as guidance for implementing a PSA Verifier that computes its results using AR4SI. The table describes which PSA Evidence claims (if any) are related to which AR4SI trustworthiness claim, and therefore what the Verifier must consider when deciding if and how to appraise a certain feature associated with the PSA Attester.

Trustworthiness Vector claims	Related PSA claims
"configuration"	Software Components (Section 4.4.1)
"executables"	ditto
"file-system"	N/A
"hardware"	Implementation ID (Section 4.2.2)
"instance-identity"	Instance ID (Section 4.2.1). The Security Lifecycle (Section 4.3.1) can also impact the derived identity.

Trustworthiness Vector claims	Related PSA claims
"runtime-opaque"	Indirectly derived from "executables", "hardware", and "instance-identity". The Security Lifecycle (Section 4.3.1) can also be relevant, e.g., any debug state will expose otherwise protected memory.
"sourced-data"	N/A
"storage-opaque"	Indirectly derived from "executables", "hardware", and "instance-identity".

Table 5: AR4SI Claims mappings

This document does not prescribe what value must be chosen based on each possible situation. When assigning specific Trustworthiness Claim values, an implementation is expected to follow the algorithm described in Section 2.3.3 of [RATS-AR4SI].

8.2. Endorsements, Reference Values, and Verification Key Material

[PSA-Endorsements] defines a protocol based on the [RATS-CoRIM] data model that can be used to convey PSA Endorsements, Reference Values, and verification key material to the Verifier.

9. Security and Privacy Considerations

This specification reuses the EAT specification and therefore the CWT specification. Hence, the security and privacy considerations of those specifications apply here as well.

Since CWTs offer different ways to protect the token, this specification profiles those options and allows signatures using public key cryptography as well as message authentication codes (MACs). COSE_Sign1 is used for digital signatures and COSE_Mac0 for MACs as defined in the COSE specification [STD96]. Note, however, that the use of MAC authentication is **NOT RECOMMENDED** due to the associated infrastructure costs for key management and protocol complexities.

A PSA Attester **MUST NOT** provide Evidence to an untrusted challenger, as it may allow attackers to interpose and trick the Verifier into believing the attacker is a legitimate Attester. This is especially relevant to protocols that use PSA attestation tokens to authenticate the attester to a Relying Party.

Attestation tokens contain information that may be unique to a device. Therefore, they may allow to single out an individual device for tracking purposes. Deployments that have privacy requirements must take appropriate measures to ensure that the token is only used to provision anonymous/pseudonym keys.

10. IANA Considerations

10.1. CBOR Web Token Claims Registration

IANA has registered the following claims in the "CBOR Web Token (CWT) Claims" registry [CWT].

10.1.1. Client ID Claim

Claim Name: psa-client-id
Claim Description: PSA Client ID
JWT Claim Name: N/A
Claim Key: 2394
Claim Value Type(s): signed integer
Change Controller: Hannes Tschofenig
Specification Document(s): Section 4.1.2 of RFC 9783

10.1.2. Security Lifecycle Claim

Claim Name: psa-security-lifecycle
Claim Description: PSA Security Lifecycle
JWT Claim Name: N/A
Claim Key: 2395
Claim Value Type(s): unsigned integer
Change Controller: Hannes Tschofenig
Specification Document(s): Section 4.3.1 of RFC 9783

10.1.3. Implementation ID Claim

Claim Name: psa-implementation-id
Claim Description: PSA Implementation ID
JWT Claim Name: N/A
Claim Key: 2396
Claim Value Type(s): byte string
Change Controller: Hannes Tschofenig
Specification Document(s): Section 4.2.2 of RFC 9783

10.1.4. Certification Reference Claim

Claim Name: psa-certification-reference
Claim Description: PSA Certification Reference
JWT Claim Name: N/A
Claim Key: 2398
Claim Value Type(s): text string
Change Controller: Hannes Tschofenig

Specification Document(s): Section 4.2.3 of RFC 9783

10.1.5. Software Components Claim

Claim Name: psa-software-components
Claim Description: PSA Software Components
JWT Claim Name: N/A
Claim Key: 2399
Claim Value Type(s): array
Change Controller: Hannes Tschofenig
Specification Document(s): Section 4.4.1 of RFC 9783

10.1.6. Verification Service Indicator Claim

Claim Name: psa-verification-service-indicator
Claim Description: PSA Verification Service Indicator
JWT Claim Name: N/A
Claim Key: 2400
Claim Value Type(s): text string
Change Controller: Hannes Tschofenig
Specification Document(s): Section 4.5.1 of RFC 9783

10.2. Media Types

This document does not register any new media types. To indicate that the transmitted content is a PSA attestation token, applications can use the `application/eat+cwt` media type defined in [EAT-MEDIATYPES] with the `eat_profile` parameter set to `tag:psacertified.org,2023:psa#tfm` (or `tag:psacertified.org,2019:psa#legacy` if the token is encoded according to the old profile; see Section 4.6).

10.3. CoAP Content-Formats Registration

IANA has registered two CoAP Content-Format IDs in the First Come First Served range of the "CoAP Content-Formats" registry [Content-Formats]:

- One for the `application/eat+cwt` media type with the `eat_profile` parameter equal to `tag:psacertified.org,2023:psa#tfm`.
- Another for the `application/eat+cwt` media type with the `eat_profile` parameter equal to `tag:psacertified.org,2019:psa#legacy`.

10.3.1. Registry Contents

Media Type: `application/eat+cwt; eat_profile="tag:psacertified.org,2023:psa#tfm"`
Encoding: -
ID: 10003
Reference: RFC 9783

Media Type: `application/eat+cwt; eat_profile="tag:psacertified.org, 2019:psa#legacy"`
Encoding: -
ID: 10004
Reference: RFC 9783

11. References

11.1. Normative References

[COSE-ALGS] Schaad, J., "CBOR Object Signing and Encryption (COSE): Initial Algorithms", RFC 9053, DOI 10.17487/RFC9053, August 2022, <https://www.rfc-editor.org/info/rfc9053>.

[CWT] IANA, "CBOR Web Token (CWT) Claims", <https://www.iana.org/assignments/cwt>.

[EAN-13] GS1, "EAN/UPC barcodes", <https://www.gs1.org/standards/barcodes/ean-upc>.

[EAT] 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/info/rfc9711>.

[EAT-MEDIATYPES] Lundblade, L., Birkholz, H., and T. Fossati, "Entity Attestation Token (EAT) Media Types", RFC 9782, DOI 10.17487/RFC9782, May 2025, <https://www.rfc-editor.org/info/rfc9782>.

[NAMED-INFO] IANA, "Named Information Hash Algorithm Registry", <https://www.iana.org/assignments/named-information>.

[PSA-Cert-Guide] PSA Certified, "PSA Certified Level 2 Step by Step Guide Version 1.1", April 2020, <https://www.psacertified.org/app/uploads/2020/07/JSADEN011-PSA_Certified_Level_2_Step-by-Step-1.1-20200403.pdf>.

[PSA-FF] Arm, "Arm PSA Firmware Framework 1.0", <https://developer.arm.com/documentation/den0063/a>.

[PSA-SM] Arm, "Platform Security Model 1.1", December 2021, <https://www.psacertified.org/app/uploads/2021/12/JSADEN014_PSA_Certified_SM_V1.1_BET0.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/info/rfc2119>.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <https://www.rfc-editor.org/info/rfc3986>.

[RFC4151] Kindberg, T. and S. Hawke, "The 'tag' URI Scheme", RFC 4151, DOI 10.17487/RFC4151, October 2005, <https://www.rfc-editor.org/info/rfc4151>.

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

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

[RFC8610] Birkholz, H., Vigano, C., and C. Bormann, "Concise Data Definition Language (CDDL): A Notational Convention to Express Concise Binary Object Representation (CBOR) and JSON Data Structures", RFC 8610, DOI 10.17487/RFC8610, June 2019, <https://www.rfc-editor.org/info/rfc8610>.

[STD94] Internet Standard 94, <https://www.rfc-editor.org/info/std94>.
At the time of writing, this STD comprises the following:

Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", STD 94, RFC 8949, DOI 10.17487/RFC8949, December 2020, <https://www.rfc-editor.org/info/rfc8949>.

[STD96] Internet Standard 96, <https://www.rfc-editor.org/info/std96>.
At the time of writing, this STD comprises the following:

Schaad, J., "CBOR Object Signing and Encryption (COSE): Structures and Process", STD 96, RFC 9052, DOI 10.17487/RFC9052, August 2022, <https://www.rfc-editor.org/info/rfc9052>.

Schaad, J., "CBOR Object Signing and Encryption (COSE): Countersignatures", STD 96, RFC 9338, DOI 10.17487/RFC9338, December 2022, <https://www.rfc-editor.org/info/rfc9338>.

[X509] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, DOI 10.17487/RFC5280, May 2008, <https://www.rfc-editor.org/info/rfc5280>.

11.2. Informative References

[Content-Formats] IANA, "CoAP Content-Formats", <https://www.iana.org/assignments/core-parameters>.

[COSE-X509] Schaad, J., "CBOR Object Signing and Encryption (COSE): Header Parameters for Carrying and Referencing X.509 Certificates", RFC 9360, DOI 10.17487/RFC9360, February 2023, <https://www.rfc-editor.org/info/rfc9360>.

[IAT-VERIFIER] Trusted Firmware, "iat-verifier", commit: 0b49b00195b7733d6fe74e8f42ed4d7b81242801, 18 August 2023, <https://git.trustedfirmware.org/plugins/gitiles/TF-M/tf-m-tools/+/refs/heads/main/iat-verifier/>.

[PSA] Arm, "Security - Platform Security Architecture", <https://developer.arm.com/documentation/101892/0100/Security---Platform-Security-Architecture?lang=en>.

[PSA-API] Arm, "PSA Certified Attestation API 1.0", October 2022, <https://arm-software.github.io/psa-api/attestation/1.0/IHI0085-PSA_Certified_Attestation_API-1.0.3.pdf>.

[PSA-Endorsements] Fossati, T., Deshpande, Y., and H. Birkholz, "A CoRIM Profile for Arm's Platform Security Architecture (PSA)", Work in Progress, Internet-Draft, draft-fdb-rats-psa-endorsements-06, 3 March 2025, <https://datatracker.ietf.org/doc/html/draft-fdb-rats-psa-endorsements-06>.

[PSA-OLD] Tschofenig, H., Frost, S., Brossard, M., Shaw, A. L., and T. Fossati, "Arm's Platform Security Architecture (PSA) Attestation Token", Work in Progress, Internet-Draft, draft-tschofenig-rats-psa-token-07, 1 February 2021, <https://datatracker.ietf.org/doc/html/draft-tschofenig-rats-psa-token-07>.

[PSACertified] PSA Certified, "PSA Certified: IoT Security Framework and Certification", <https://psacertified.org>.

[RATS-AR4SI] Voit, E., Birkholz, H., Hardjono, T., Fossati, T., and V. Scarlata, "Attestation Results for Secure Interactions", Work in Progress, Internet-Draft, draft-ietf-rats-ar4si-08, 6 February 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-rats-ar4si-08>.

[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-07, 3 March 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-rats-corim-07>.

[RATS-QWESTOKEN] Mandyam, G., Sekhar, V., and S. Mohammed, "The Qualcomm Wireless Edge Services (QWES) Attestation Token", Work in Progress, Internet-Draft, draft-mandyam-rats-qwestoken-00, 1 November 2019, <https://datatracker.ietf.org/doc/html/draft-mandyam-rats-qwestoken-00>.

[RATS-TDX] Kostal, G., Dittakavi, S., Yeluri, R., Xia, H., and J. Yu, "EAT profile for Intel(r) Trust Domain Extensions (TDX) attestation result", Work in Progress, Internet-Draft, draft-kdyxy-rats-tdx-eat-profile-02, 13 December 2024, <https://datatracker.ietf.org/doc/html/draft-kdyxy-rats-tdx-eat-profile-02>.

[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/info/rfc9334>.

[TF-M] Trusted Firmware, "Trusted Firmware-M", <https://www.trustedfirmware.org/projects/tf-m/>.

[TLS12-IoT] Tschofenig, H., Ed. and T. Fossati, "Transport Layer Security (TLS) / Datagram Transport Layer Security (DTLS) Profiles for the Internet of Things", RFC 7925, DOI 10.17487/RFC7925, July 2016, <https://www.rfc-editor.org/info/rfc7925>.

[TLS13-IoT] Tschofenig, H., Fossati, T., and M. Richardson, "TLS/DTLS 1.3 Profiles for the Internet of Things", Work in Progress, Internet-Draft, draft-ietf-uta-tls13-iot-profile-14, 5 May 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-uta-tls13-iot-profile-14>.

Appendix A. Examples

The following examples show PSA attestation tokens for an hypothetical system comprising a single measured software component. The attesting device is in a lifecycle state (Section 4.3.1) of SECURED. The attestation has been requested from a client residing in the SPE.

The example in Appendix A.1 illustrates the case where the IAK is an asymmetric key. A COSE Sign1 envelope is used to wrap the PSA-token claims set.

Appendix A.2 illustrates the case where the IAK is a symmetric key and a COSE Mac0 envelope is used instead.

The claims sets are identical, except for the Instance ID which is synthesized from the key material.

The examples have been created using the `iat-verifier` tool [IAT-VERIFIER].

A.1. COSE Sign1 Token

```
{
  / ueid /                      256: h'010202020202020202020202
02020202020202020202020202020202020202',
  / psa-implementation-id /   2396: h'000000000000000000000000
00000000000000000000000000000000000000',
  / eat_nonce /                10: h'010101010101010101010101
01010101010101010101010101010101010101',
  / psa-client-id /           2394: 2147483647,
  / psa-security-lifecycle /  2395: 12288,
  / eat_profile /              265: "tag:psacertified.org,2023:psa#tfm",
  / bootseed /                 268: h'0000000000000000',
  / psa-software-components / 2399: [
    {
      / signer ID /         5: h'04040404040404040404040404040
4040404040404040404040404040404',
      / measurement value / 2: h'03030303030303030303030303030
3030303030303030303030303030303',
      / measurement type /  1: "PRoT"
    }
  ]
}
```

The JWK representation of the IAK used for creating the COSE Sign1 signature over the PSA token is:

```
{
  "kty": "EC",
  "crv": "P-256",
  "alg": "ES256",
  "x": "Tl4iCZ47zrRbRG0TVf0dw7VFlHtv18HInYhnmMNybo8",
  "y": "gNcLhAslaqw0pi7eEEM2TwRAlfADR0uR4Bggkq-xPy4",
  "d": "Q__-y5X4CFp8QOHT6nkL7063jN131YUDpkwWAPkbM-c"
}
```

The resulting COSE object is:

```
18([
  h'A10126',
  {},
  h'A819010058210102020202020202020202020202020202020202020202
02020202020202020219095C582000000000000000000000000000000000
00000000000000000000000000000A582001010101010101010101010101
01010101010101010101010101010119095A1A7FFFFFFF19095B19300019
01097821746167 3A7073616365727469666965642E6F72672C323032333A7073
612374666D19010C4800000000000000019095F81A3055820040404040404
04040404040404040404040404040404040404040404040402582003030303
03030303030303030303030303030303030303030303030303030301645052
6F54',
  h'786E937A4C42667AF3847399319CA95C7E7DBABDC9B50FDB8DE3F6BFF4AB
82FF80C42140E2A488000219E3E10663193DA69C75F52B798EA10B2F7041A90E
8E5A'
])
```

which has the following base16 encoding:

```
d28443a10126a0590100a819010058210102020202020202020202020202
02020202020202020202020202020202020219095c5820000000000000
00000000000000000000000000000000000000000000000000a582001
01010101010101010101010101010101010101010101010101010101
0119095a1a7fffffff19095b193000190109782174616 73a707361636572
7469666965642e6f72672c323032333a7073612374666d19010c48000000
000000000019095f81a3055820040404040404040404040404040404040
04040404040404040404040404040258200303030303030303030303030
03030303030303030303030303030303030303030303016450526f545840786e
937a4c42667af3847399319ca95c7e7dbabdc9b50fdb8de3f6bff4ab82ff
80c42140e2a488000219e3e10663193da69c75f52b798ea10b2f7041a90e
8e5a
```

A.2. COSE Mac0 Token

```
{
  / ueid /                      256: h'01C557BD4FADC83F756FCA2CD5
EA2DCC8B82159BB4E7453D6A744D4EECD6D0AC60',
  / psa-implementation-id /   2396: h'000000000000000000000000000
0000000000000000000000000000000000000',
  / eat_nonce /                10: h'010101010101010101010101
0101010101010101010101010101010101',
  / psa-client-id /           2394: 2147483647,
  / psa-security-lifecycle /  2395: 12288,
  / eat_profile /              265: "tag:psacertified.org,2023:psa#tfm",
  / psa-boot-seed /            268: h'0000000000000000',
  / psa-software-components / 2399: [
    {
      / signer ID /         5: h'040404040404040404040404040404040
4040404040404040404040404040404',
      / measurement value / 2: h'030303030303030303030303030303030
3030303030303030303030303030303',
      / measurement type /  1: "PRoT"
    }
  ]
}
```

The JWK representation of the IAK used for creating the COSE Mac0 signature over the PSA token is:

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

{
  "kty": "oct",
  "alg": "HS256",
  "k": "3gOLNKyhJXaMXjNXq40Gs2e5qw1-i-Ek7cpH_gM6W7epPTB_8imqNv8k\
       \bBKVlk-s9xq3qm7E_WECt7OYMlWtkg"
}
```

The resulting COSE object is:

```
17([
  h'A10105',
  {},
  h'A8190100582101C557BD4FADC83F756FCA2CD5EA2DCC8B82159BB4E7453D
6A744D4EECD6D0AC6019095C5820000000000000000000000000000000000000
0000000000000000000000000000000A582001010101010101010101010101010
0101010101010101010101010101010119095A1A7FFFFFFF19095B19300019
01097821746167 3A707361636572746966696 5642E6F72672C323032333A7073
612374666D19010C4800000000000000019095F81A3055820040404040404040
0404040404040404040404040404040404040404040404040402582003030303
0303030303030303030303030303030303030303030303030301645052
6F54',
  h'CF88D330E7A5366A95CF744A4DBF0D50304D405EDD8B2530E243EDDBD317
7820'
])
```

which has the following base16 encoding:

```
d18443a10105a0590100a8190100582101c557bd4fadc83f756fca2cd5ea
2dcc8b82159bb4e7453d6a744d4eecd6d0ac6019095c5820000000000000
0000000000000000000000000000000000000000000000000000a582001
0101010101010101010101010101010101010101010101010101010101
0119095a1a7fffffff19095b193000190109782174 61673a707361636572
7469666965642e6f72672c323032333a7073612374666d19010c48000000
000000000019095f81a3055820040404040404040404040404040404040
0404040404040404040404040404025820030303030303030303030303
0303030303030303030303030303030303030303030303016450526f545820cf88
d330e7a5366a95cf744a4dbf0d50304d405edd8b2530e243eddbd3177820
```

Acknowledgments

Thank you Carsten Bormann for help with the CDDL. Thanks to Nicholas Wood, Eliot Lear, Yaron Sheffer, Kathleen Moriarty, and Ned Smith for ideas, comments, and suggestions.

Contributors

Laurence Lundblade
Security Theory LLC
Email: lgl@securitytheory.com

Tamas Ban
Arm Limited
Email: Tamas.Ban@arm.com

Sergei Trofimov
Arm Limited
Email: Sergei.Trofimov@arm.com

Authors' Addresses

Hannes Tschofenig
University of Applied Sciences Bonn-Rhein-Sieg
Germany
Email: Hannes.Tschofenig@gmx.net

Simon Frost
Arm Limited
Email: Simon.Frost@arm.com

Mathias Brossard
Arm Limited
Email: Mathias.Brossard@arm.com

Adrian Shaw
HP Labs
Email: adrianlshaw@acm.org

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