

RATS
Internet-Draft
Intended status: Informational
Expires: 8 September 2022

H. Tschofenig
S. Frost
M. Brossard
Arm Limited
A. Shaw
HP Labs
T. Fossati
Arm Limited
7 March 2022

Arm's Platform Security Architecture (PSA) Attestation Token
draft-tschofenig-rats-psa-token-09

Abstract

The Platform Security Architecture (PSA) is a family of hardware and firmware security specifications, as well as open-source reference implementations, to help device makers and chip manufacturers build best-practice security into products. Devices that are PSA compliant are able to produce attestation tokens as described in this memo, which are the basis for a number of different protocols, including secure provisioning and network access control. This document specifies the PSA attestation token structure and semantics.

The PSA attestation token is a profiled Entity Attestation Token (EAT).

This specification describes what claims are used in an attestation token generated by PSA compliant systems, how these claims get serialized to the wire, and how they are cryptographically protected.

Note to Readers

Source for this draft and an issue tracker can be found at <https://github.com/thomas-fossati/draft-psa-token> (<https://github.com/thomas-fossati/draft-psa-token>).

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of [BCP 78](#) and [BCP 79](#).

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

Internet-Draft

PSA Attestation Token

March 2022

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 8 September 2022.

Copyright Notice

Copyright (c) 2022 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to [BCP 78](#) and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document.

Table of Contents

1.	Introduction	3
2.	Conventions and Definitions	3
2.1.	Glossary	4
3.	PSA Claims	4
3.1.	Caller Claims	4
3.1.1.	Nonce	4
3.1.2.	Client ID	5
3.2.	Target Identification Claims	5
3.2.1.	Instance ID	5
3.2.2.	Implementation ID	6
3.2.3.	Certification Reference	6
3.3.	Target State Claims	7
3.3.1.	Security Lifecycle	7
3.3.2.	Boot Seed	8
3.4.	Software Inventory Claims	8
3.4.1.	Software Components	8
3.5.	Verification Claims	10
3.5.1.	Verification Service Indicator	10
3.5.2.	Profile Definition	11
4.	Backwards Compatibility Considerations	11
5.	Token Encoding and Signing	12
6.	Freshness Model	13
7.	Collated CDDL	13

8.	Implementation Status	16
9.	Security and Privacy Considerations	16
10.	Verification	16
11.	IANA Considerations	17
11.1.	CBOR Web Token Claims Registration	17

11.1.1.	Client ID Claim	17
11.1.2.	Security Lifecycle Claim	18
11.1.3.	Implementation ID Claim	18
11.1.4.	Boot Seed Claim	18
11.1.5.	Certification Reference Claim	19
11.1.6.	Software Components Claim	19
11.1.7.	Verification Service Indicator Claim	19
11.2.	Media Type Registration	20
11.3.	CoAP Content-Formats Registration	21
11.3.1.	Registry Contents	21
12.	References	21
12.1.	Normative References	21
12.2.	Informative References	22
Appendix A.	Example	23
	Contributors	25
	Acknowledgments	25
	Authors' Addresses	25

[1.](#) Introduction

Trusted execution environments are now present in many devices, which provide a safe environment to place 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. Various APIs have been developed by Arm as part of the Platform Security Architecture [[PSA](#)] framework. This document focuses on the output provided by PSA's Initial Attestation API. Since the 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

[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.

[2.1.](#) Glossary

RoT Root of Trust, 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 Root of Trust 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.

SPE Secure Processing Environment, 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 Trusted Execution Environment (TEE), or "secure world".)

NSPE Non Secure Processing Environment, the security domain outside of the SPE, the Application domain, typically containing the application firmware, operating systems, and general hardware. (Equivalent to Rich Execution Environment (REE), or "normal world".)

[3.](#) PSA Claims

This section describes the claims to be used in a PSA attestation token.

CDDL [[RFC8610](#)] along with text descriptions is used to define each

claim independent of encoding. The following CDDL type(s) are reused by different claims:

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

[3.1.](#) Caller Claims

[3.1.1.](#) Nonce

The Nonce claim is used to carry the challenge provided by the caller to demonstrate freshness of the generated token.

The EAT [[I-D.ietf-rats-eat](#)] nonce (claim key 10) is used. The following constraints apply to the nonce-type:

- * The length MUST be either 32, 48, or 64 bytes.
- * Only a single nonce value is conveyed. Per [[I-D.ietf-rats-eat](#)] the array notation is not used for encoding the nonce value.

This claim MUST be present in a PSA attestation token.

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

[3.1.2.](#) 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  
)
```

[3.2.](#) Target Identification Claims

[3.2.1.](#) Instance ID

The Instance ID claim represents the unique identifier of the Initial Attestation Key (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-bytes key hash.

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  
)
```

[3.2.2.](#) Implementation ID

The Implementation ID claim uniquely identifies the implementation 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 3.2.1](#).

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

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

[3.2.3](#). 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. It MUST be represented as a thirteen-digit [\[EAN-13\]](#).

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.

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

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

[3.3](#). Target State Claims

3.3.1. 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:

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

The PSA lifecycle states are illustrated in Figure 1. 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.

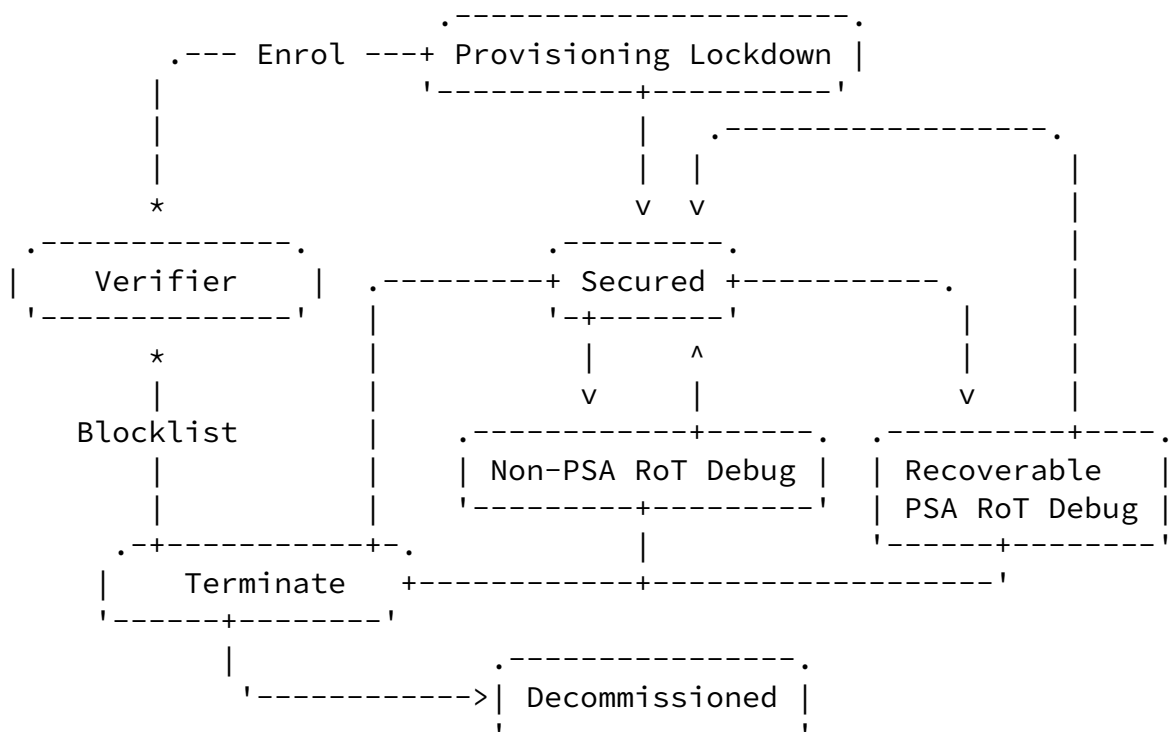


Figure 1: PSA Lifecycle States


```

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
)

```

[3.3.2.](#) Boot Seed

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

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

```

psa-boot-seed-type = bytes .size 32

```

```

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

```

[3.4.](#) Software Inventory Claims

[3.4.1.](#) Software Components

The Software Components claim is a list of software components that includes all the software loaded by the PSA RoT. This claim **SHALL** 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, as described in [[I-D.ietf-rats-architecture](#)], a relying party will typically see the result of the verification process from the Verifier in form of an attestation result, rather than the "naked" PSA token from the attesting endpoint. 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 endorsements 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 = {  
  ? 1 => text,           ; measurement type  
  ? 2 => psa-hash-type,   ; measurement value  
  ? 4 => text,           ; version  
  ? 5 => psa-hash-type,   ; signer id  
  ? 6 => text,           ; measurement description  
}  
  
psa-software-components = (  
  psa-software-components-key => [ + psa-software-component ]  
)
```

[3.4.1.1](#). Measurement Type

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

The following measurement types MAY be used:

- * "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

[3.4.1.2.](#) 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.

[3.4.1.3.](#) 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.

[3.4.1.4.](#) Signer ID

The Signer ID attribute (key=5) is the hash of a signing authority public key for the software component. 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 [\[PSA-SM\]](#).

[3.4.1.5.](#) 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 [\[IANA-HashFunctionTextualNames\]](#).

[3.5.](#) Verification Claims

[3.5.1.](#) Verification Service Indicator

The Verification Service Indicator claim is a hint used by a relying party to locate a validation service for the token. The value is a text string that can be used to locate the service or a URL

specifying the address of the service. A Verifier 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  
)
```

[3.5.2.](#) 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 profile (claim key 265) is used. The following constraints apply to its type:

- * The URI encoding MUST be used.
- * The value MUST be <http://arm.com/psa/2.0.0>.

This claim MUST be present in a PSA attestation token.

See [Section 4](#), for considerations about backwards compatibility with previous versions of the PSA attestation token format.

```
psa-profile-type = "http://arm.com/psa/2.0.0"
```

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

[4.](#) Backwards Compatibility Considerations

A previous version of this specification (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 retired and their use is deprecated.

Table 1 provides the mappings between the deprecated and new claim keys.

+=====+		
	PSA_IOT_PROFILE_1	http://arm.com/psa/2.0.0
+=====+		
Nonce	-75008	10 (EAT nonce)
+-----+		
Instance ID	-75009	256 (EAT euid)
+-----+		
Profile	-75000	265 (EAT eat_profile)
Definition		
+-----+		
Client ID	-75001	2394
+-----+		
Security	-75002	2395
Lifecycle		
+-----+		
Implementation ID	-75003	2396
+-----+		
Boot Seed	-75004	2397
+-----+		
Certification	-75005	2398
Reference		
+-----+		
Software	-75006	2399
Components		
+-----+		

Verification	-75010	2400	
Service Indicator			
+-----+	+-----+	+-----+	+-----+

Table 1: Claim key mappings

Unless compatibility with existing infrastructure is a concern, emitters (e.g., devices that implement the PSA Attestation API) SHOULD produce tokens with the claim keys specified in this document.

To simplify the transition to the token format described in this document it is RECOMMENDED that receivers (e.g., PSA Attestation Verifiers) accept tokens encoded according to the old profile (PSA_IOT_PROFILE_1) as well as to the new profile (<http://arm.com/psa/2.0.0>), at least for the time needed to their clients to upgrade.

5. Token Encoding and Signing

The PSA attestation token is encoded in CBOR [[RFC8949](#)] format. Only definite-length string, arrays, and maps are allowed.

Cryptographic protection is obtained by wrapping the psa-token map in a COSE Web Token (CWT) [[RFC8392](#)]. For asymmetric key algorithms, the signature structure MUST be COSE_Sign1. For symmetric key algorithms, the signature structure MUST be COSE_Mac0.

Acknowledging the variety of markets, regulations and use cases in which the PSA attestation token can be used, this specification does not impose any strong requirement on the cryptographic algorithms that need to be supported by Attesters and Verifiers. It is assumed that some form of out-of-band discovery and negotiation is in place to allow interoperability between the involved parties, and that the flexibility provided by the COSE format is sufficient to deal with the level of cryptographic agility needed to adapt to specific use cases.

The CWT CBOR tag (61) is not used. An application that needs to exchange PSA attestation tokens can wrap the serialised COSE_Sign1 or COSE_Mac0 in the media type defined in [Section 11.2](#) or the CoAP

Content-Format defined in [Section 11.3](#).

[6](#). Freshness Model

The PSA Token supports the freshness models for attestation Evidence based on nonces and epoch handles ([Section 10.2](#) and 10.3 of [\[I-D.ietf-rats-architecture\]](#)) 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.

[7](#). 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-boot-seed-key = 2397  
psa-certification-reference-key = 2398
```

```
psa-software-components-key = 2399  
psa-verification-service-indicator-key = 2400
```

```
; from EAT  
nonce-label = 10  
ueid-label = 256  
profile-label = 265
```

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

```
psa-boot-seed-type = bytes .size 32
```

```

psa-boot-seed = (
    psa-boot-seed-key => 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}"

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

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

```

```

psa-profile-type = "http://arm.com/psa/2.0.0"

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 = {
    ? 1 => text,           ; measurement type
    2 => psa-hash-type,    ; measurement value
    ? 4 => text,           ; version
    5 => psa-hash-type,    ; signer id
    ? 6 => text,           ; measurement description
}

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
)

```

[8.](#) Implementation Status

Independent implementations of this specification are provided by the Trusted Firmware project [[TF-M](#)], the Veraison project [[Veraison](#)], and Xclaim [[Xclaim](#)]. All three implementations are released as open-source software.

[9.](#) Security and Privacy Considerations

This specification re-uses the CWT and the EAT 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 based on use of public key cryptography as well as MAC authentication. The token MUST be signed following the structure of the COSE specification [[RFC8152](#)]. The COSE type MUST be COSE_Sign1 for public key signatures or COSE_Mac0 for MAC authentication. Note however that use of MAC authentication is NOT RECOMMENDED due to the associated infrastructure costs for key management and protocol complexities. It may also restrict the ability to interoperate with third parties.

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

[10.](#) Verification

To verify the token, the primary need is to check correct encoding and signing as detailed in [Section 5](#). In particular, the Instance ID claim is used (together with the kid in the COSE header, if present) to assist in locating the public key used to verify the signature covering the CWT token. The key used for verification is supplied to the Verifier by an authorized Endorser along with the corresponding Attester's Instance ID.

In addition, the Verifier will typically operate a policy where values of some of the claims in this profile can be compared to reference values, registered with the Verifier for a given deployment, in order 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.

The protocol used to convey Endorsements and Reference Values to the Verifier is not in scope for this document.

[11.](#) IANA Considerations

[11.1.](#) CBOR Web Token Claims Registration

This specification requests IANA to register the following claims in

the "CBOR Web Token (CWT) Claims" registry [[IANA-CWT](#)].

[11.1.1.1.](#) Client ID Claim

- * Claim Name: psa-client-id
- * Claim Description: PSA Client ID

Tschofenig, et al.

Expires 8 September 2022

[Page 17]

Internet-Draft

PSA Attestation Token

March 2022

- * JWT Claim Name: N/A
- * Claim Key: TBD (requested value: 2394)
- * Claim Value Type(s): signed integer
- * Change Controller: [[Authors of this RFC]]
- * Specification Document(s): [Section 3.1.2](#) of [[this RFC]]

[11.1.1.2.](#) Security Lifecycle Claim

- * Claim Name: psa-security-lifecycle
- * Claim Description: PSA Security Lifecycle
- * JWT Claim Name: N/A
- * Claim Key: TBD (requested value: 2395)
- * Claim Value Type(s): unsigned integer
- * Change Controller: [[Authors of this RFC]]
- * Specification Document(s): [Section 3.3.1](#) of [[this RFC]]

[11.1.1.3.](#) Implementation ID Claim

- * Claim Name: psa-implementation-id
- * Claim Description: PSA Implementation ID
- * JWT Claim Name: N/A

- * Claim Key: TBD (requested value: 2396)
- * Claim Value Type(s): byte string
- * Change Controller: [[Authors of this RFC]]
- * Specification Document(s): [Section 3.2.2](#) of [[this RFC]]

[11.1.4.](#) Boot Seed Claim

- * Claim Name: psa-boot-seed
- * Claim Description: PSA Boot Seed

Tschofenig, et al. Expires 8 September 2022 [Page 18]

Internet-Draft PSA Attestation Token March 2022

- * JWT Claim Name: N/A
- * Claim Key: TBD (requested value: 2397)
- * Claim Value Type(s): byte string
- * Change Controller: [[Authors of this RFC]]
- * Specification Document(s): [Section 3.3.2](#) of [[this RFC]]

[11.1.5.](#) Certification Reference Claim

- * Claim Name: psa-certification-reference
- * Claim Description: PSA Certification Reference
- * JWT Claim Name: N/A
- * Claim Key: TBD (requested value: 2398)
- * Claim Value Type(s): text string
- * Change Controller: [[Authors of this RFC]]
- * Specification Document(s): [Section 3.2.3](#) of [[this RFC]]

[11.1.6.](#) Software Components Claim

- * Claim Name: psa-software-components
- * Claim Description: PSA Software Components
- * JWT Claim Name: N/A
- * Claim Key: TBD (requested value: 2399)
- * Claim Value Type(s): array
- * Change Controller: [[Authors of this RFC]]
- * Specification Document(s): [Section 3.4.1](#) of [[this RFC]]

[11.1.7.](#) Verification Service Indicator Claim

- * Claim Name: psa-verification-service-indicator
- * Claim Description: PSA Verification Service Indicator

- * JWT Claim Name: N/A
- * Claim Key: TBD (requested value: 2400)
- * Claim Value Type(s): text string
- * Change Controller: [[Authors of this RFC]]
- * Specification Document(s): [Section 3.5.1](#) of [[this RFC]]

[11.2.](#) Media Type Registration

IANA is requested to register the "application/psa-attestation-token" media type [\[RFC2046\]](#) in the "Media Types" registry [\[IANA-MediaTypes\]](#) in the manner described in [RFC 6838](#) [\[RFC6838\]](#), which can be used to indicate that the content is a PSA Attestation Token.

- * Type name: application

- * Subtype name: psa-attestation-token
- * Required parameters: n/a
- * Optional parameters: n/a
- * Encoding considerations: binary
- * Security considerations: See the Security Considerations section of [[this RFC]]
- * Interoperability considerations: n/a
- * Published specification: [[this RFC]]
- * Applications that use this media type: Attesters and Relying Parties sending PSA attestation tokens over HTTP(S), CoAP(S), and other transports.
- * Fragment identifier considerations: n/a
- * Additional information:
 - Magic number(s): n/a
 - File extension(s): n/a
 - Macintosh file type code(s): n/a

- * Person & email address to contact for further information: Hannes Tschofenig, Hannes.Tschofenig@arm.com
- * Intended usage: COMMON
- * Restrictions on usage: none
- * Author: Hannes Tschofenig, Hannes.Tschofenig@arm.com
- * Change controller: IESG
- * Provisional registration? No

[11.3.](#) CoAP Content-Formats Registration

IANA is requested to register the CoAP Content-Format ID for the "application/psa-attestation-token" media type in the "CoAP Content-Formats" registry [[IANA-CoAP-Content-Formats](#)].

[11.3.1.](#) Registry Contents

- * Media Type: application/psa-attestation-token
- * Encoding: -
- * Id: [[To-be-assigned by IANA]]
- * Reference: [[this RFC]]

[12.](#) References

[12.1.](#) Normative References

- [EAN-13] GS1, "International Article Number - EAN/UPC barcodes", 2019, <<https://www.gs1.org/standards/barcodes/ean-upc>>.
- [I-D.ietf-rats-eat] Lundblade, L., Mandyam, G., and J. O'Donoghue, "The Entity Attestation Token (EAT)", Work in Progress, Internet-Draft, [draft-ietf-rats-eat-12](#), 24 February 2022, <<https://datatracker.ietf.org/doc/html/draft-ietf-rats-eat-12>>.
- [PSA-FF] Arm, "Platform Security Architecture Firmware Framework 1.0 (PSA-FF)", February 2019, <https://developer.arm.com/-/media/Files/pdf/PlatformSecurityArchitecture/Architect/DEN0063-PSA_Firmware_Framework-1.0.0-2.pdf>.

- [PSA-SM] Arm, "Platform Security Architecture Security Model 1.0 (PSA-SM)", February 2019, <https://developer.arm.com/-/media/Files/pdf/PlatformSecurityArchitecture/Architect/DEN0079_PSA_SM_ALPHA-03_RC01.pdf>.

- [RFC2046] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types", [RFC 2046](#), DOI 10.17487/RFC2046, November 1996, <<https://www.rfc-editor.org/rfc/rfc2046>>.
- [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>>.
- [RFC6838] Freed, N., Klensin, J., and T. Hansen, "Media Type Specifications and Registration Procedures", [BCP 13](#), [RFC 6838](#), DOI 10.17487/RFC6838, January 2013, <<https://www.rfc-editor.org/rfc/rfc6838>>.
- [RFC8152] Schaad, J., "CBOR Object Signing and Encryption (COSE)", [RFC 8152](#), DOI 10.17487/RFC8152, July 2017, <<https://www.rfc-editor.org/rfc/rfc8152>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in [RFC 2119](#) Key Words", [BCP 14](#), [RFC 8174](#), DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/rfc/rfc8174>>.
- [RFC8392] Jones, M., Wahlstroem, E., Erdtman, S., and H. Tschofenig, "CBOR Web Token (CWT)", [RFC 8392](#), DOI 10.17487/RFC8392, May 2018, <<https://www.rfc-editor.org/rfc/rfc8392>>.
- [RFC8610] Birkholz, H., Vigano, C., and C. Bormann, "Concise Data Definition Language (CDDL): A Notational Convention to Express Concise Binary Object Representation (CBOR) and JSON Data Structures", [RFC 8610](#), DOI 10.17487/RFC8610, June 2019, <<https://www.rfc-editor.org/rfc/rfc8610>>.
- [RFC8949] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", STD 94, [RFC 8949](#), DOI 10.17487/RFC8949, December 2020, <<https://www.rfc-editor.org/rfc/rfc8949>>.

[12.2](#). Informative References

- [I-D.ietf-rats-architecture]
Birkholz, H., Thaler, D., Richardson, M., Smith, N., and W. Pan, "Remote Attestation Procedures Architecture", Work

in Progress, Internet-Draft, [draft-ietf-rats-architecture-15](https://datatracker.ietf.org/doc/html/draft-ietf-rats-architecture-15), 8 February 2022,
<<https://datatracker.ietf.org/doc/html/draft-ietf-rats-architecture-15>>.

[IANA-CoAP-Content-Formats]

IANA, "CoAP Content-Formats", 2022,
<<https://www.iana.org/assignments/core-parameters>>.

[IANA-CWT] IANA, "CBOR Web Token (CWT) Claims", 2022,

<<https://www.iana.org/assignments/cwt/cwt.xhtml#claims-registry>>.

[IANA-HashFunctionTextualNames]

IANA, "Hash Function Textual Names", 2022,
<<https://www.iana.org/assignments/hash-function-text-names>>.

[IANA-MediaTypes]

IANA, "Media Types", 2022,
<<http://www.iana.org/assignments/media-types>>.

[PSA]

Arm, "Platform Security Architecture Resources", 2022,
<<https://developer.arm.com/architectures/security-architectures/platform-security-architecture/documentation>>.

[PSACertified]

PSA Certified, "PSA Certified IoT Security Framework", 2022, <<https://psacertified.org>>.

[TF-M]

Linaro, "Trusted Firmware-M", 2022,
<<https://www.trustedfirmware.org/projects/tf-m/>>.

[Veraison]

The Veraison Project, "Veraison psatoken package", 2022,
<<https://github.com/veraison/psatoken>>.

[Xclaim]

Lundblade, L., "Xclaim", 2022,
<<https://github.com/laurencelundblade/xclaim>>.

[Appendix A](#). Example

The following example shows a PSA attestation token for an hypothetical system comprising two measured software components (a boot loader and a trusted RTOS). The attesting device is in a lifecycle state [Section 3.3.1](#) of SECURED. The attestation has been requested from a client residing in the SPE:

Internet-Draft

PSA Attestation Token

March 2022

```

{
  / eat_profile /          265: "http://arm.com/psa/2.0.0",
  / psa-client-id /        2394: 1,
  / psa-lifecycle /        2395: 12288,
  / psa-implementation-id / 2396: h'50515253545556575051
52535455565750515253545556575051525354555657',
  / psa-boot-seed /        2397: h'DEADBEEFDEADBEEFDEAD
BEEFDEADBEEFDEADBEEFDEADBEEFDEADBEEFDEADBEEF',
  / psa-certification-reference / 2398: "1234567890123",
  / psa-software-components / 2399: [
    {
      / measurement type / 1: "BL",
      / measurement value / 2: h'0001020400010204000102040001020
400010204000102040001020400010204',
      / signer ID /        5: h'519200FF519200FF519200FF519200F
F519200FF519200FF519200FF519200FF'
    },
    {
      / measurement type / 1: "PRoT",
      / measurement value / 2: h'0506070805060708050607080506070
805060708050607080506070805060708',
      / signer ID /        5: h'519200FF519200FF519200FF519200F
F519200FF519200FF519200FF519200FF'
    }
  ],
  / nonce /                10: h'00010203000102030001020300010203
00010203000102030001020300010203',
  / ueid /                  256: h'01A0A1A2A3A0A1A2A3A0A1A2A3A0A1A2
A3A0A1A2A3A0A1A2A3A0A1A2A3A0A1A2A3',
  / psa-verification-service-indicator / 2400: "https://psa-ve
rifier.org"
}

```

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

```

{
  "kty": "EC",
  "crv": "P-256",
  "x": "MKBCTNIcKUSDii11ySs3526iDZ8AiTo7Tu6KPAqv7D4",
  "y": "4Etl6SRW2YiLUrN5vfvVHuhp7x8PxltmWWlbbM4IFyM",
  "d": "870MB6gfuTJ4HtUnUvYMyJpr5eUZNP4Bk43bVdj3eAE",

```

```
"use": "enc",
"kid": "1"
}
```

The resulting COSE object is:

```
18(
  [
    / protected / h'A10126',
    / unprotected / {},
    / payload / h'AA1901097818687474703A2F2F61726D2E636F6D2F
7073612F322E302E3019095A0119095B19300019095C58205051525354555657
50515253545556575051525354555657505152535455565719095D5820DEADBE
EFDEADBEEFDEADBEEFDEADBEEFDEADBEEFDEADBEEFDEADBEEFDEADBEEF19095E
6D3132333435363738393031323319095F82A30162424C025820000102040001
0204000102040001020400010204000102040001020400010204055820519200
FF519200FF519200FF519200FF519200FF519200FF519200FF519200FFA30164
50526F5402582005060708050607080506070805060708050607080506070805
06070805060708055820519200FF519200FF519200FF519200FF519200FF5192
00FF519200FF519200FF0A582000010203000102030001020300010203000102
03000102030001020300010203190100582101A0A1A2A3A0A1A2A3A0A1A2A3A0
A1A2A3A0A1A2A3A0A1A2A3A0A1A2A3A0A1A2A3190960781868747470733A2F2F
7073612D76657269666965722E6F7267',
    / signature / h'E3B80C143403ECB744B1D6EF732872A1A3E682783E
939F72A3CEF6BF74EF4BC5E7065725FF5C948770B673C5896D3F796F55D144FC
B456BEA832EB13E8258DB8'
  ]
)
```

Contributors

We would like to thank the following colleagues for their contributions:

* Laurence Lundblade
Security Theory LLC
lgl@securitytheory.com

* Tamas Ban
Arm Limited
Tamas.Ban@arm.com

* Sergei Trofimov
Arm Limited
Sergei.Trofimov@arm.com

Acknowledgments

Thanks to Carsten Bormann for help with the CDDL and Nicholas Wood for ideas and comments.

Authors' Addresses

Tschofenig, et al. Expires 8 September 2022 [Page 25]

Internet-Draft PSA Attestation Token March 2022

Hannes Tschofenig
Arm Limited
Email: Hannes.Tschofenig@arm.com

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

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

Adrian Shaw
HP Labs
Email: Adrian.Shaw@hp.com

Thomas Fossati
Arm Limited
Email: Thomas.Fossati@arm.com

