PSA Attestation API 1.0

Document number:	ARM IHI 0085
Release Quality:	Release
Issue Number:	2
Confidentiality:	Non-Confidential
Date of Issue:	19/02/2020

Contents

About this document

Release Information

The change history table lists the changes that have been made to this document. See the Document History section for a detailed list of changes.

Date	Version	Confidentiality	Change
Feb 2019	*1.0 beta 0*	Non-Confidential	Initial publication.
June 2019	1.0.0	Non-Confidential	First stable release Uses the PSA common error status codes. Modified the API parameters to align with other PSA APIs. Updated the claims and lifecycle to match the latest PSA Security Model. Updated CBOR example in the appendix.
Aug 2019	1.0.1	Non-Confidential	Recommend type byte 0x01 for arm_psa_UEID. Remove erroneous guidance regarding EAT's origination claim.
Feb 2020	1.0.2	Non-Confidential	Clarify the claim number of Instance ID Permit COSE-Mac0 for signing tokens (with appropriate warning) Update URLs

PSA Attestation API 1.0

Arm Non-Confidential Document Licence ("Licence")

connection with the Document or any products based thereon. Visit Arm's website at https://www.arm.com/company/policies/trademarks for more information about Arm's trademarks.

The validity, construction and performance of this Licence shall be governed by English Law.

Arm Limited. Company 02557590 registered in England.
110 Fulbourn Road, Cambridge, England CB1 9NJ.

Arm document reference: LES-PRE-21585

References

This document refers to the following documents.

Ref	Document Number	Title
ARM DEN 0079	PSA Security Model (PSA-SM)	https://developer.arm.com/architectures/security-architectures/platform-security-architecture
ARM DEN 0063	PSA Firmware Framework (PSA-FF)	https://developer.arm.com/architectures/security-architectures/platform-security-architecture
N/A	IETF Entity Attestation Token (EAT) draft	https://tools.ietf.org/html/draft-ietf-rats-eat-02
RFC 7049	IETF Concise Binary Object Representation (CBOR)	https://tools.ietf.org/html/rfc7049
RFC 8152	CBOR Object Signing and Encryption (COSE)	https://tools.ietf.org/html/rfc8152
EAN-13	International Article Number	https://www.gs1.org/standards/barcodes/ean-upc

Terms and abbreviations

This document uses the following terms and abbreviations.

Term	Meaning
CBOR	Concise Binary Object Representation
EAT	Entity Attestation Token
IAK	Initial Attestation Key
PSA	Platform Security Architecture

Potential for change

The contents of this specification are subject to change.

In particular, the following may change:

- Feature addition, modification, or removal
- Parameter addition, modification, or removal
- Numerical values, encodings, bit maps

Conventions

Typographical conventions

The typographical conventions are:

italic

Introduces special terminology, and denotes citations.

bold

Denotes signal names, and is used for terms in descriptive lists, where appropriate.

`monospace`

Used for assembler syntax descriptions, pseudocode, and source code examples.

Also used in the main text for instruction mnemonics and for references to other items appearing in assembler syntax descriptions, pseudocode, and source code examples.

SMALL CAPITALS

Used for some common terms such as IMPLEMENTATION DEFINED.

Used for a few terms that have specific technical meanings, and are included in the Glossary.

Red text

Indicates an open issue.

Blue text

Indicates a link. This can be

- A cross-reference to another location within the document
- A URL, for example http://infocenter.arm.com

Numbers

Numbers are normally written in decimal. Binary numbers are preceded by `0b`, and hexadecimal numbers by `0x`.

In both cases, the prefix and the associated value are written in a monospace font, for example `0xFFFF0000`. To improve readability, long numbers can be written with an underscore separator between every four characters, for example `0xFFFF_0000_0000_0000`. Ignore any underscores when interpreting the value of a number.

Pseudocode descriptions

This book uses a form of pseudocode to provide precise descriptions of the specified functionality. This pseudocode

is written in a monospace font. The pseudocode language is described in the Arm Architecture Reference Manual.

Assembler syntax descriptions

This book is not expected to contain assembler code or pseudo code examples.

Any code examples are shown in a `monospace` font.

Current status and anticipated changes

First draft, major changes and re-writes to be expected.

Feedback

Arm welcomes feedback on its documentation.

Feedback on this book

If you have comments on the content of this book, send an e-mail to arm.psa-feedback@arm.com. Give:

- The title (PSA Attestation API).
- The number and issue (ARM IHI 0085 1.0 Release 2)
- The page numbers to which your comments apply.
- The rule identifiers to which your comments apply, if applicable.
- A concise explanation of your comments.

Arm also welcomes general suggestions for additions and improvements.

1 Introduction

Arm's Platform Security Architecture (PSA) is a holistic set of threat models, security analyses, hardware and firmware architecture specifications, and an open source firmware reference implementation. PSA provides a recipe, based on industry best practice, that allows security to be consistently designed in, at both a hardware and firmware level.

The PSA Attestation API is a standard interface provided by the PSA Root of Trust. The definition of the PSA Root of Trust is described in the PSA Security Model (PSA-SM).

This document includes:

- a set of common use cases
- information about the attestation report and the format
- the associated Application Programming Interface (API)

The API can be used either to directly sign data or as a way to bootstrap trust in other attestation schemes. PSA provides a framework and the minimal generic security features allowing OEM and service providers to integrate various attestation schemes on top of the PSA Root of Trust.

2 Use cases and rationale

The following subsections describe the primary use cases that PSA aims to support in this version of the API. Other use cases are also possible.

The PSA Root of Trust reports information, known as claims, that can be used to determine the exact implementation of the PSA Root of Trust and its security state. If the PSA Root of Trust loads other components, then it also includes information about what it has loaded. Other components outside of the PSA Root of Trust can add additional information to the report by calling the provided API, which will include and sign the additional information. The PSA Root of Trust signs attestation reports using the Initial Attestation Key (IAK).

2.1 Device enrolment

Enrolment is the ability for an online service to enlist a device. For example, a generic connected sensor that becomes part of a company's deployment. As part of the enrolment process, credentials need to be created for each device. However, the devices themselves need to be trustworthy to ensure that credentials are not leaked.

A common solution to this problem is to certify security hardware using third-party labs, who are trusted to deliver worthwhile certifications. By placing trust in evaluation reports (such as Common Criteria or PSA Certified), one can ascertain whether a Root of Trust exhibits important security properties. For example, one important property is the ability to generate a key pair of good quality (using a non-predictable random number generator) and store it in a tamper-proof area, which provides strong assurance that a device private key is only ever known by that device. Each device instance contains a protected attestation key that can be used to prove that they are a particular certified implementation.

During such an enrolment process, a device might generate a new key pair and create a Certificate Signing Request (CSR) or equivalent, containing:

- The public key of the generated key-pair.
- A proof of possession of the corresponding private key (in general this is the public key signed by the private key). This protects against man-in-the-middle attacks where an attacker can hijack the enrolment to insert their own public key into the device request.
- An initial attestation, in order for the recipient to assess how that particular combination of hardware and firmware can be trusted.

The CSR is then passed to a Certification Authority who can assign it an identity with the new service and then return an identity certificate signed using the private key of the Certification Authority. The Certification Authority may be operated by the company who owns the devices or operated by a trusted third party. Creating extra identities on devices is expected to be a routine operation.

If a device is built with PSA isolation Level 3, where all applications inside a device execute inside their own Secure Partition, then it allows several mutually-distrustful providers to install their applications side-by-side without having to worry about leaking assets from one Secure Partition to another.

The attestation identity can be verified in an attestation process and checked against certification information. At the end of the process the verifier can establish a secure connection to the attested endpoint and deliver credentials. For example, they may be service access credentials.

2.2 Identifying certification

The combination of a hardware entity and the software installed on that entity can be certified to conform to some published security level.

Manufacturers of devices can advertise a security certification as an incentive to purchase their devices. To gain the certification a manufacturer can engage a test lab to verify the hardware and software combination of a device conforms to specific standards. Certification should not be declared by the device, instead it is a dynamic situation where the hardware and software state can be checked against the current known certification status for that combination.

The initial attestation report declares the state of the device to a verification service. The verification service can then:

- Verify the production status of the device identity. For example, to identify whether the device is in an inventory and whether it is a secured production device or a development device).
- Verify the certification status of a device. This involves checking that all components up to date, correctly signed, and certified to work together.
- Complete an attestation protocol to establish a secure connection that is bound to the device identity.

2.3 Integrity reporting

A party may want to check the received list of claims against a database of known measurements for each component in order to decide which level of trust should be applied. Additional information can be included, such as the version numbers for all software running on the device. As a minimum, the device provides a hash for each loaded component. Boot measurements are included in order to assess if there are obvious signs of tampering with the device firmware.

Initial attestation requires three services:

- Enrolment verification service enforcing policy as part of service enrolment of the device.
- Production verification service (OEM), providing the production state of an attestation identity
- Certification verification service (third party), verifying that all attested components are up to date, signed correctly, and certified to work together.

It is possible to further separate these roles. For example, there may be a separate software verification service.

These services can be hosted by different parties in online or offline settings:

- The first service requires generating a challenge, reading back the device's token, and validating the signature of the token.
- The second service may periodically log the current security state for all addressable devices and make that information available upon request. It does not require the knowledge of any pre-shared secret or a prior trust exchange with a device vendor. The various databases required for the full verification process may be local, replicated, or centralized, depending on the particular market.

Further information about using existing attestation protocols can be found in the PSA-SM.

3 PSA Initial Attestation report

This section begins with a description of the information model for the report and then describes the expected format.

3.1 Information model

The following table describes the mandatory and optional claims in the report:

Claim	Mandatory	Description
Auth challenge	Yes	Input object from the caller. For example, this can be a cryptographic nonce or a hash of locally attested data. The length must be 32, 48, or 64 bytes.
Instance ID	Yes	Represents the unique identifier of the instance. It is a hash of the public key corresponding to the Initial Attestation Key (IAK). If the IAK is a symmetric key then the Instance ID is a hash of the IAK. The full definition is in the PSA-SM.
Verification service indicator	No	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.
Profile definition	No	Contains the name of a document that describes the 'profile' of the report. The document name may include versioning. The value for this specification is **PSA_IOT_PROFILE_1**.
Implementation ID	Yes	Uniquely identifies the underlying immutable PSA RoT. A verification service can use this claim to locate the details of the verification process. Such details include the implementation's origin and associated certification state. The full definition is in the PSA-SM.
Client ID	Yes	Represents the Partition ID of the caller. It is 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. The full definition of a Partition ID is provided by the PSA Firmware Framework (PSA-FF). It is essential that this claim is checked in the verification process to ensure that a security domain cannot spoof a report from another security domain.

Security Lifecycle	Yes	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 • version[7:0] – IMPLEMENTATION DEFINED state. The PSA security lifecycle states consist of the following values: • PSA_LIFECYCLE_UNKNOWN (0x0000u) • PSA_LIFECYCLE_ASSEMBLY_AND_TEST (0x1000u) • PSA_LIFECYCLE_PSA_ROT_PROVISIONING (0x2000u) • PSA_LIFECYCLE_SECURED (0x3000u) • PSA_LIFECYCLE_NON_PSA_ROT_DEBUG (0x4000u) • PSA_LIFECYCLE_RECOVERABLE_PSA_ROT_DEBUG (0x5000u) • PSA_LIFECYCLE_DECOMMISSIONED (0x6000u) For PSA, a remote verifier can only trust reports from the PSA RoT when it is in SECURED or NON_PSA_ROT_DEBUG major states.
Hardware version	No	Provides metadata linking the token to the GDSII that went to fabrication for this instance. It can be used to link the class of chip and PSA RoT to the data on a certification website. It must be represented as a thirteen-digit EAN-13
Boot seed	Yes	Represents a random value created at system boot time that can allow differentiation of reports from different boot sessions.
Software components	Yes (unless the No Software Measurements claim is specified)	A list of software components that represent all the software loaded by the PSA Root of Trust. This claim is needed for the rules outlined in the PSA-SM. Each entry has the following fields: 1. Measurement type 2. Measurement value 3. Version 4. Signer ID 5. Measurement description The full definition of the software component is described in Software Components This claim is required to be compliant with the PSA-SM.
No Software Measurements	Yes (if no software components specified)	In the event that the implementation does not contain any software measurements then the Software Components claim above can be omitted but instead it is mandatory to include this claim to indicate this is a deliberate state.

	This claim is intended for devices that are not compliant with the PSA-SM.

3.1.1 Software components

Each software component in the Software Components claim must include the required properties of the following table:

Key ID	Type	Required	Description
1	Measurement type	No	A short string representing the role of this software component (e.g. 'BL' for boot loader). Expected types may include: • BL (a bootloader) • 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)
2	Measurement value	Yes	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.
3	Reserved	No	Reserved
4	Version	No	The issued software version in the form of a text string. The value of this claim corresponds to the entry in the original signed manifest of the component. This field must be present to be compliant with the PSA-SM.
5	Signer ID	No	The hash of a signing authority public key for the software component. The value of this claim corresponds 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 field must be present to be compliant with the PSA-SM.
6	Measurement description	No	Description of the software component, which represents the way in which the measurement value of the software component is computed. The value is a text string containing an abbreviated description (or name) of the measurement method which can be used to lookup the details of the method in a profile document. This claim may normally be excluded, unless there is an exception to the default measurement described in the profile for a specific component.

3.2 Report format and signing

This section describes the specific representation, encoding and signing of the information described in the Information Model.

3.2.1 Token encoding

The report is represented as a token, which must be formatted in accordance to the IETF Entity Attestation Token (EAT) draft specification. The token consists of a series of claims declaring evidence as to the nature of the instance of hardware and software. The claims are encoded with the CBOR format.

3.2.2 Signing

The token is signed following the structure of the CBOR Object Signing and Encryption (COSE) specification:

- For asymmetric key algorithms, the signature structure must be COSE-Sign1. An asymmetric key algorithm is needed to achieve all the use cases defined in Use cases and rationale.
- For symmetric key algorithms, the structure must be COSE-Mac0.

Warning

A symmetric key is **strongly discouraged** due to the associated infrastructure costs for key management and operational complexities. It may also restrict the ability to interoperate with scenarios that involve third parties (see Use cases and rationale).

3.2.3 EAT standard claims

The token is modelled to include custom values that correspond to the following EAT standard claims (as expressed in the draft EAT proposal):

- **nonce** (mandatory); `arm_psa_nonce` is used instead
- **UEID** (mandatory); `arm_psa_UEID` is used instead

A future version of the profile, corresponding to an issued standard, might declare support for both custom and standard claims as a transitionary state towards exclusive use of standard claims.

3.2.4 EAT custom claims

The token can include the following EAT custom claims. PSA custom claims have a root identity of -75000.

Some fields must be at least 32 bytes to provide sufficient cryptographic strength.

Key ID	Type	Name	CBOR type
-75000	Profile Definition	`arm_psa_profile_id`	Text string
-75001	Client ID	`arm_psa_partition_id`	Unsigned integer or Negative integer
-75002	Security Lifecycle	`arm_psa_security_lifecycle`	Unsigned integer
-75003	Implementation ID	`arm_psa_implementation_id`	Byte string (>=32 bytes)
-75004	Boot seed	`arm_psa_boot_seed`	Byte string (>=32 bytes)
-75005	Hardware version	`arm_psa_hw_version`	Text string

-75006	Software components (compound map claim)	`arm_psa_sw_components`	Array of map entries. Each map entry must have the following types for each Key-Value: 1. Text string (type) 2. Byte string (measurement, >=32 bytes) 3. Reserved 4. Text string (version) 5. Byte string (signer ID, >=32 bytes) 6. Text string (measurement description)
-75007	No software measurements	`arm_psa_no_sw_measurements`	Unsigned integer
-75008	Auth challenge	`arm_psa_nonce`	Byte string
-75009	Instance ID	`arm_psa_UEID`	Byte string (the type byte should be set to `0x01`. The type byte is described in the (EAT) draft.)
-75010	Verification service indicator	`arm_psa_origination`	Byte string

An example report can be found in Example Report

4 API reference

The API has a respective header file that must be provided by the implementation. The named header must be **<psa/initial_attestation.h>**.

All the following definitions must be present in the header file.

All the functions are defined in the C language. The APIs make use of standard 'C' data types as defined in the ISO C99 specification.

4.1 Error handling

All functions must return a status indication of type `psa_status_t`, which is defined by `<psa/error.h>`. The definition of `<psa/error.h>` is provided by (PSA-FF). This is an enumeration of integer values, with `0` (`PSA_SUCCESS`) indicating successful operation and negative values indicating errors.

Each API documents the specific error codes that might be returned, and the meaning of each error.

All parameters of pointer type must be valid, non-null pointers unless the pointer is to a buffer of length 0 or the function's documentation explicitly describes the behavior when the pointer is null. For implementations where a null pointer dereference usually aborts the application, passing NULL as a function parameter where a null pointer is not allowed should abort the caller in the habitual manner.

Pointers to input parameters may be in read-only memory. Output parameters must be in writable memory. Output parameters that are not buffers must also be readable, and the implementation must be able to write to a non-buffer output parameter and read back the same value.

4.2 General definitions

4.2.1 PSA_INITIAL_ATTEST_API_VERSION_MAJOR

```
#define PSA_INITIAL_ATTEST_API_VERSION_MAJOR  (1)
```

The major version of this implementation of the PSA Attestation API.

4.2.2 PSA_INITIAL_ATTEST_API_VERSION_MINOR

```
#define PSA_INITIAL_ATTEST_API_VERSION_MINOR  (0)
```

The minor version of this implementation of the PSA Attestation API.

4.2.3 PSA_INITIAL_ATTEST_MAX_TOKEN_SIZE

```
#define PSA_INITIAL_ATTEST_MAX_TOKEN_SIZE  /*...*/
```

The maximum possible size of a token. The value of this constant is IMPLEMENTATION DEFINED.

4.3 Challenge sizes

The following constants define the valid challenge sizes that must be supported by the function psa_initial_attest_get_token() and psa_initial_attest_get_token_size().

An implementation must not support other challenge sizes.

4.3.1 PSA_INITIAL_ATTEST_CHALLENGE_SIZE_32

```
#define PSA_INITIAL_ATTEST_CHALLENGE_SIZE_32  (32u)
```

A challenge size of 32 bytes (256 bits).

4.3.2 PSA_INITIAL_ATTEST_CHALLENGE_SIZE_48

```
#define PSA_INITIAL_ATTEST_CHALLENGE_SIZE_48  (48u)
```

A challenge size of 48 bytes (384 bits).

4.3.3 PSA_INITIAL_ATTEST_CHALLENGE_SIZE_64

```
#define PSA_INITIAL_ATTEST_CHALLENGE_SIZE_64  (64u)
```

A challenge size of 64 bytes (512 bits).

4.4 Attestation

4.4.1 psa_initial_attest_get_token

Retrieve the Initial Attestation Token.

```
psa_status_t psa_initial_attest_get_token(const uint8_t *auth_challenge,
                                          size_t challenge_size,
                                          uint8_t *token_buf,
                                          size_t token_buf_size,
                                          size_t *token_size);
```

Parameters

`auth_challenge`	Buffer with a challenge object. The challenge object is data provided by the caller. For example, it may be a cryptographic nonce or a hash of data (such as an external object record). If a hash of data is provided then it is the caller's responsibility to ensure that the data is protected against replay attacks (for example, by including a cryptographic nonce within the data).
`challenge_size`	Size of the buffer `auth_challenge` in bytes. The size must always be a supported challenge size. Supported challenge sizes are defined in the Challenge Sizes section.
`token_buf`	Output buffer where the attestation token is to be written.
`token_buf_size`	Size of `token_buf`. The expected size can be determined by using the `psa_initial_attest_get_token_size` function.
`token_size`	Output variable for the actual token size.

Outputs

`*token_buf`	On success, the attestation token.
`*token_size`	On success, the number of bytes written into `token_buf`.

Returns: `psa_status_t`

`PSA_SUCCESS`	Action was performed successfully.
`PSA_ERROR_SERVICE_FAILURE`	The implementation failed to fully initialize.
`PSA_ERROR_BUFFER_TOO_SMALL`	`token_buf` is too small for the attestation token.
`PSA_ERROR_INVALID_ARGUMENT`	The challenge size is not supported.
`PSA_ERROR_GENERIC_ERROR`	An unspecified internal error has occurred.

Description

Retrieves the Initial Attestation Token. A challenge can be passed as an input to mitigate replay attacks.

4.4.2 `psa_initial_attest_get_token_size`

Calculate the size of an Initial Attestation Token.

```
psa_status_t psa_initial_attest_get_token_size(size_t challenge_size,
                                               size_t *token_size);
```

Parameters

`challenge_size`	Size of a challenge object in bytes. This must be a supported challenge size as specified in the Challenge Sizes section.
`token_size`	Output variable for the token size.

Outputs

`*token_size`	On success, the maximum size of an attestation token in bytes when using the specified `challenge_size`

Returns: `psa_status_t`

`PSA_SUCCESS`	Action was performed successfully.
`PSA_ERROR_SERVICE_FAILURE`	The implementation failed to fully initialize.

PSA_ERROR_INVALID_ARGUMENT	The challenge size is not supported.
PSA_ERROR_GENERIC_ERROR	An unspecified internal error has occurred.

Description

Retrieve the exact size of the Initial Attestation Token in bytes, given a specific challenge size.

5 Appendix: Example report

An example report is included here in extended CBOR diagnostic form for illustrative purposes:

```
18(
[
/ protected / h'a10126' / {
    \ alg \ 1: -7 \ ECDSA 256 \
  } / ,
/ unprotected / {},
/ payload / h'a93a000124fb582000010203040506070809 0a0b0c0d0e0f1011121
31415161718191a1b1c1d1e1f3a000124fa582000010203040506070809 0a0b0c0d0e
0f101112131415161718191a1b1c1d1e1f3a000124fd84a402582000010203040506 0
708090a0b0c0d0e0f101112131415161718191a1b1c1d1e1f0465332e312e34055820
000102030405060708090a0b0c0d0e0f101112131415161718191a1b1c1d1e1f01624
24ca402582000010203040506070809 0a0b0c0d0e0f101112131415161718191a1b1c
1d1e1f0463312e3105582000010203040506070809 0a0b0c0d0e0f101112131415161
718191a1b1c1d1e1f016450526f54a402582000010203040506070809 0a0b0c0d0e0f
101112131415161718191a1b1c1d1e1f0463312e3005582000010203040506070809
0a0b0c0d0e0f101112131415161718191a1b1c1d1e1f016441526f54a4025820000102
030405060708090a0b0c0d0e0f101112131415161718191a1b1c1d1e1f0463322e320
558200001020304050607080 90a0b0c0d0e0f101112131415161718191a1b1c1d1e1f
01634170703a000124f91930003a000124ff582000010203040506070809 0a0b0c0d0
e0f101112131415161718191a1b1c1d1e1f3a000125016c7073615f76657269666965
723a000124f8203a000125005821010001020304050607080 90a0b0c0d0e0f1011121
31415161718191a1b1c1d1e1f3a000124f7715053415f496f545f50524f46494c455f
31' / {
   / arm_psa_boot_seed / -75004: h'000102030405060708090a0b0c0d0e0f10
   111213141516171819 1a1b1c1d1e1f',
   / arm_psa_implementation_id / -75003: h'000102030405060708090a0b0c
   0d0e0f101112131415161718191a1b1c1d1e1f',
   / arm_psa_sw_components / -75006: [
        {
          / measurement / 2: h'000102030405060708090a0b0c0d0e0f101112
          1314151617181 91a1b1c1d1e1f',
          / version / 4: "3.1.4",
          / signerID / 5: h'000102030405060708090a0b0c0d0e0f101112131
          415161718191a1b1c1d1e1f',
          / type / 1: "BL"
        },
        {
          / measurement / 2: h'000102030405060708090a0b0c0d0e0f101112
          1314151617181 91a1b1c1d1e1f',
          / version / 4: "1.1",
          / signerID / 5: h'000102030405060708090a0b0c0d0e0f101112131
          415161718191a1b1c1d1e1f',
          / type / 1: "PRoT"
        },
        {
          / measurement / 2: h'000102030405060708090a0b0c0d0e0f101112
          1314151617181 91a1b1c1d1e1f',
          / version / 4: "1.0",
          / signerID / 5: h'000102030405060708090a0b0c0d0e0f101112131
          415161718191a1b1c1d1e1f',
```

```
          / type / 1: "ARoT"
        },
        {
          / measurement / 2: h'000102030405060708090a0b0c0d0e0f101112
          131415161718191a1b1c1d1e1f',
          / version / 4: "2.2",
          / signerID / 5: h'000102030405060708090a0b0c0d0e0f101112131
          415161718191a1b1c1d1e1f',
          / type / 1: "App"
        }
      ],
    / arm_psa_security_lifecycle / -75002: 12288 / SECURED /,
    / arm_psa_nonce / -75008: h'000102030405060708090a0b0c0d0e0f10111
    2131415161718191a1b1c1d1e1f',
    / arm_psa_origination / -75010: "psa_verifier",
    / arm_psa_partition_id / -75001: -1,
    / arm_psa_UEID / -75009: h'010001020304050607080090a0b0c0d0e0f1011
    12131415161718191a1b1c1d1e1f',
    / arm_psa_profile_id / -75000: "PSA_IoT_PROFILE_1"
  }),
  } / ,
/ signature / h'58860508ee7e8cc48eba872dbb5d694a542b1322ad0d51023c197
0df429f06501c683a95108a0cced0a6e80e0966f22bd63d1c0056974a11ba332d7877
87fb4f'
]
)
```

6 Document history

Date	Changes
2019-02-25	*1.0 beta 0*
2019-06-12	*1.0 Release 0 (1.0.0)* • The API functions now use PSA's common `psa_status_t` return type. • Error values now use standard PSA error codes, which are now defined in `<psa/error.h>`. • Input parameters are now separate from output parameters. There are no longer any • in/out parameters. • Size types have been replaced with `size_t` instead of `uint32_t`. • Some parameter names have been changed to improve legibility. • The description of the Implementation ID claim has been rewritten to better match the definition in PSA-SM. • Signer ID is no longer a mandatory part of the Software Components claim. However, it is needed for PSA-SM compliance.

	• Explicitly describe which optional claims are required for PSA-SM compliance. • Added lifecycle state (`PSA_LIFECYCLE_ASSEMBLY_AND_TEST`). • Clarifications and improvements to the description of some API elements and to the structure of the document. • Updated CBOR example in the appendix
2019-08-30	*1.0 Release 1 (1.0.1)* • Fixed typos and descriptions based on feedback. • Recommend type byte 0x01 for `arm_psa_UEID`. • Remove erroneous guidance regarding EAT's origination claim – it should not be used to find a verification service.
2020-02-20	*1.0 Release 2 (1.0.2)* • Clarify the claim number of Instance ID • Permit COSE-Mac0 for signing tokens (with appropriate warning) • Update URLs