Workgroup: RATS Working Group
Internet-Draft: draft-ietf-rats-eat-15
Published: 30 September 2022
Intended Status: Standards Track
Expires: 3 April 2023
Authors: L. Lundblade
Security Theory LLC
G. Mandyam
Qualcomm Technologies Inc.
J. O'Donoghue
Qualcomm Technologies Inc.
C. Wallace
Red Hound Software, Inc.

The Entity Attestation Token (EAT)

Abstract

An Entity Attestation Token (EAT) provides an attested claims set that describes state and characteristics of an entity, a device like a smartphone, IoT device, network equipment or such. This claims set is used by a relying party, server or service to determine how much it wishes to trust the entity.

An EAT is either a CBOR Web Token (CWT) or JSON Web Token (JWT) with attestation-oriented claims.

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-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 3 April 2023.

Copyright Notice

Table of Contents

1. Introduction

An Entity Attestation Token (EAT) is a message or token made up of claims about an entity. An entity may be a device, some hardware or some software. The claims are ultimately used by a relying party who decides if and how it will interact with the entity. The relying party may choose to trust, not trust or partially trust the entity. For example, partial trust may be allowing a monetary transaction only up to a limit.

The security model and goal for attestation are unique and are not the same as for other security standards like those for server authentication, user authentication and secured messaging. The reader is assumed to be familiar with the goals and security model for attestation as described in [RATS.Architecture].

This document defines some common claims that are potentially of broad use. EAT additionally allows proprietary claims and for further claims to be standardized. Here are some examples:

- Make and model of manufactured consumer device
- Make and model of a chip or processor, particularly for a security-oriented chip
- Identification and measurement of the software running on a device
- Configuration and state of a device
- Environmental characteristics of a device like its GPS location
- Formal certifications received

EAT is constructed to support a wide range of use cases.

No single set of claims can accommodate all use cases so EAT is constructed as a framework for defining specific attestation tokens for specific use cases. In particular, EAT provides a profile mechanism to be able to clearly specify the claims needed, the cryptographic algorithms that should be used and other for a particular token and use case.

The entity side of an EAT implementation generates the claims and typically signs them with an attestation key. It is responsible for protecting the attestation key. Some EAT implementations will use components with very high resistance to attack like TPMs or secure elements. Other may rely solely on simple SW defenses.

Nesting of tokens and claims sets is accommodated for composite devices that have multiple subsystems.

An EAT may be encoded in either JSON [RFC8259] or CBOR [RFC8949] as needed for each use case. EAT is built on CBOR Web Token (CWT) [RFC8392] and JSON Web Token (JWT) [RFC7519] and inherits all their characteristics and their security mechanisms.

1.1. Entity Overview

The document uses the term "entity" to refer to the target of an EAT. Many of the claims defined in this document are claims about an entity, which is equivalent to an attesting environment as defined in [RATS.Architecture]. An entity may be the whole device, a subsystem, a subsystem of a subsystem, etc. Correspondingly, the EAT format allows claims to be organized using mechanisms like submodules and nested EATs (see Section 4.2.18). The entity to which a claim applies is the submodule in which it appears, or to the top-level entity if it doesn't appear in a submodule.

An entity also corresponds to a "system component", as defined in the Internet Security Glossary [RFC4949]. That glossary also defines "entity" and "system entity" as something that may be a person or organization as well as a system component. In the EAT context, "entity" never refers to a person or organization. The hardware and software that implement a server or service used by a web site may be an entity, but the organization that runs the web site is not an entity nor is the web site itself. An entity is an implementation in hardware, software or both.

Some examples of entities:

- A Secure Element
- A TEE
- A card in a network router
- A network router, perhaps with each card in the router a submodule
- An IoT device
- An individual process
- An app on a smartphone
- A smartphone with many submodules for its many subsystems
- A subsystem in a smartphone like the modem or the camera

An entity may have strong security defenses against hardware invasive attacks. It may also have low security, having no special security defenses. There is no minimum security requirement to be an entity.

1.2. EAT as a Framework

EAT is a framework for defining attestation tokens for specific use cases, not a specific token definition. While EAT is based on and compatible with CWT and JWT, it can also be described as:

- An identification and type system for claims in claims-sets
- Definitions of common attestation-oriented claims
- Claims are defined in CDDL and serialized using CBOR or JSON
- Security envelopes based on COSE and JOSE
- Nesting of claims sets and tokens to represent complex and compound devices
- A profile mechanism for specifying and identifying specific token formats for specific use cases

EAT uses the name/value pairs the same as CWT and JWT to identify individual claims. Section 4 defines common attestation-oriented claims that are added to the CWT and JWT IANA registries. As with CWT and JWT, no claims are mandatory and claims not recognized should be ignored.

Unlike, but compatible with CWT and JWT, EAT defines claims using Concise Data Definition Language (CDDL) [RFC8610]. In most cases the same CDDL definition is used for both the CBOR/CWT serialization and the JSON/JWT serialization.

Like CWT and JWT, EAT uses COSE and JOSE to provide authenticity, integrity and optionally confidentiality. EAT places no new restrictions on cryptographic algorithms, retaining all the cryptographic flexibility of CWT, COSE, JWT and JOSE.

EAT defines a means for nesting tokens and claims sets to accommodate composite devices that have multiple subsystems and multiple attesters. Full tokens with security envelopes may be embedded in an enclosing token. The nested token and the enclosing token do not have to use the same encoding (e.g., a CWT may be enclosed in a JWT).

EAT adds the ability to detach claims sets and send them separately from a security enveloped EAT that contains a digest of the detached claims set.

This document registers no media or content types for the identification of the type of EAT, its serialization format or security envelope. That is left for a follow-on document.

Finally, the notion of an EAT profile is introduced that facilitates the creation of narrowed definitions of EAT tokens for specific use cases in follow-on documents.

1.3. Operating Model and RATS Architecture

The EAT format follows the operational model described in Figure 1 in [RATS.Architecture]. To summarize, an attester generates evidence in the form of a claims set describing various characteristics of an entity. Evidence is usually signed by a key that proves the attester and the evidence it produces are authentic. The claims set includes a nonce or some other means to assure freshness.

A verifier confirms an EAT is valid by verifying the signature and may vet some claims using reference values. The verifier then produces attestation results, which may also be represented as an EAT. The attestation results are provided to the relying party, which is the ultimate consumer of the Remote Attestation Procedure. The relying party uses the attestation results as needed for its use case, perhaps allowing an entity to access a network, allowing a financial transaction or such. In some cases, the verifier and relying party are not distinct entities.

1.3.1. Relationship between Evidence and Attestation Results

Any claim defined in this document or in the IANA CWT or JWT registry may be used in evidence or attestation results. The relationship of claims in attestation results to evidence is fundamentally governed by the verifier and the verifier's policy.

A common use case is for the verifier and its policy to perform checks, calculations and processing with evidence as the input to produce a summary result in attestation results that indicates the overall health and status of the entity. For example, measurements in evidence may be compared to reference values the results of which are represented as a simple pass/fail in attestation results.

It is also possible that some claims in the Evidence will be forwarded unmodified to the relying party in attestation results. This forwarding is subject to the verifier's implementation and policy. The relying party should be aware of the verifier's policy to know what checks it has performed on claims it forwards.

The verifier may modify claims it forwards, for example, to implement a privacy preservation functionality. It is also possible the verifier will put claims in the attestation results that give details about the entity that it has computed or looked up in a database. For example, the verifier may be able to put an "oemid" claim in the attestation results by performing a look up based on a UEID (serial number) it received in evidence.

This specification does not establish any normative rules for the verifier to follow, as these are a matter of local policy. It is up to each relying party to understand the processing rules of each verifier to know how to interpret claims in attestation results.

2. Terminology

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.

This document reuses terminology from JWT [RFC7519] and CWT [RFC8392].

Claim: A piece of information asserted about a subject. A claim is represented as pair with a value and either a name or key to identify it.

Claim Name: A unique text string that identifies the claim. It is used as the claim name for JSON encoding.

Claim Key: The CBOR map key used to identify a claim.

Claim Value: The value portion of the claim. A claim value can be any CBOR data item or JSON value.

Claims Set: The CBOR map or JSON object that contains the claims conveyed by the CWT or JWT.

This document reuses terminology from RATS Architecure [RATS.Architecture]

Attester: A role performed by an entity (typically a device) whose evidence must be appraised in order to infer the extent to which the attester is considered trustworthy, such as when deciding whether it is authorized to perform some operation.

Verifier: A role that appraises the validity of evidence about an attester and produces attestation results to be used by a relying party.

Relying Party: A role that depends on the validity of information about an attester, for purposes of reliably applying application specific actions. Compare /relying party/ in [RFC4949].

Evidence: A set of claims generated by an attester to be appraised by a verifier. Evidence may include configuration data, measurements, telemetry, or inferences.

Attestation Results: The output generated by a verifier, typically including information about an attester, where the verifier vouches for the validity of the results

Reference Values: A set of values against which values of claims can be compared as part of applying an appraisal policy for evidence. Reference Values are sometimes referred to in other documents as known-good values, golden measurements, or nominal values, although those terms typically assume comparison for equality, whereas here reference values might be more general and be used in any sort of comparison.

Endorsement: A secure statement that an Endorser vouches for the integrity of an attester's various capabilities such as claims collection and evidence signing.

3. Top-Level Token Definition

An EAT is a "message", a "token", or such whose content is a Claims-Set about an entity or some number of entities. An EAT MUST always contains a Claims-Set.

Authenticity and integrity protection MUST be provided for EATs. This document relies on CWT or JWT for this purpose. Extensions to this specification MAY use other methods of protection.

The identification of a protocol element as an EAT follows the general conventions used for CWTs and JWTs. Identification depends on the protocol carrying the EAT. In some cases it may be by media type (e.g., in a HTTP Content-Type field). In other cases it may be through use of CBOR tags. There is no fixed mechanism across all use cases.

This document also defines a new top-level message, the detached EAT bundle (see Section 5), which holds a collection of detached claims sets and an EAT that provides integrity and authenticity protection for them. Detached EAT bundles can be either CBOR or JSON encoded.

The following CDDL defines the top-level `$$EAT-CBOR-Tagged-Token`, `$$EAT-CBOR-Untagged-Token` and `$$EAT-JSON-Token-Formats` sockets, enabling future token formats to be defined. Any new format that plugs into one or more of these sockets MUST be defined by an IETF standards action. Of particular use may be a token type that provides no direct authenticity or integrity protection for use with transports mechanisms that do provide the necessary security services [UCCS].

Nesting of EATs is allowed and defined in Section 4.2.18.1.2. This includes the nesting of an EAT that is a different format than the enclosing EAT. The definition of Nested-Token references the CDDL defined in this section. When new token formats are defined, the means for identification in a nested token MUST also be defined.

```
EAT-CBOR-Token = $$EAT-CBOR-Tagged-Token / $$EAT-CBOR-Untagged-Token

$$EAT-CBOR-Tagged-Token /= CWT-Tagged-Message
$$EAT-CBOR-Tagged-Token /= BUNDLE-Tagged-Message

$$EAT-CBOR-Untagged-Token /= CWT-Untagged-Message
$$EAT-CBOR-Untagged-Token /= BUNDLE-Untagged-Message
```

```
EAT-JSON-Token = $$EAT-JSON-Token-Formats

$$EAT-JSON-Token-Formats /= JWT-Message
$$EAT-JSON-Token-Formats /= BUNDLE-Untagged-Message
```

4. The Claims

This section describes new claims defined for attestation that are to be added to the CWT [IANA.CWT.Claims] and JWT [IANA.JWT.Claims] IANA registries.

All definitions, requirements, creation and validation procedures, security considerations, IANA registrations and so on from CWT and JWT carry over to EAT.

This section also describes how several extant CWT and JWT claims apply in EAT.

The set of claims that an EAT must contain to be considered valid is context dependent and is outside the scope of this specification. Specific applications of EATs will require implementations to understand and process some claims in particular ways. However, in the absence of such requirements, all claims that are not understood by implementations MUST be ignored.

CDDL, along with a text description, is used to define each claim independent of encoding. Each claim is defined as a CDDL group. In Section 7 on encoding, the CDDL groups turn into CBOR map entries and JSON name/value pairs.

Each claim defined in this document is added to the `$$Claims-Set-Claims` socket group. Claims defined by other specifications MUST also be added to the `$$Claims-Set-Claims` socket group.

All claims in an EAT MUST use the same encoding except where otherwise explicitly stated (e.g., in a CBOR-encoded token, all claims must be CBOR-encoded).

This specification includes a CDDL definition of most of what is defined in [RFC8392]. Similarly, this specification includes CDDL for most of what is defined in [RFC7519]. These definitions are in Appendix D and are not normative.

Each claim described has a unique text string and integer that identifies it. CBOR-encoded tokens MUST use only the integer for claim keys. JSON-encoded tokens MUST use only the text string for claim names.

4.1. eat_nonce (EAT Nonce) Claim

An EAT nonce is either a byte or text string or an array of byte or text strings. The array option supports multistage EAT verification and consumption.

A claim named "nonce" was defined and registered with IANA for JWT, but MUST NOT be used because it does not support multiple nonces. No previous "nonce" claim was defined for CWT. To distinguish from the previously defined JWT "nonce" claim, this claim is named "eat_nonce" in JSON-encoded EATs. The CWT nonce defined here is intended for general purpose use and retains the "Nonce" claim name instead of an EAT-specific name.

An EAT nonce MUST have at least 64 bits of entropy. A maximum EAT nonce size is set to limit the memory required for an implementation. All receivers MUST be able to accommodate the maximum size.

In CBOR, an EAT nonce is a byte string. The minimum size is 8 bytes. The maximum size is 64 bytes.

In JSON, an EAT nonce is a text string. It is assumed that only characters represented by the lower 7 bits of each byte will be used, so the text string must be one-seventh longer because the 8th bit doesn't contribute to entropy. The minimum size for JSON-encoded EATs is 10 bytes and the maximum size is 74 bytes.

```
$$Claims-Set-Claims //=
    (nonce-label => nonce-type / [ 2* nonce-type ])

nonce-type = JC< tstr .size (10..74), bstr .size (8..64)>
```

4.2. Claims Describing the Entity

The claims in this section describe the entity itself. They describe the entity whether they occur in evidence or occur in attestation results. See Section 1.3.1 for discussion on how attestation results relate to evidence.

4.2.1. ueid (Universal Entity ID) Claim

The "ueid" claim conveys a UEID, which identifies an individual manufactured entity like a mobile phone, a water meter, a Bluetooth speaker or a networked security camera. It may identify the entire entity or a submodule. It does not identify types, models or classes of entities. It is akin to a serial number, though it does not have to be sequential.

UEIDs MUST be universally and globally unique across manufacturers and countries. UEIDs MUST also be unique across protocols and systems, as tokens are intended to be embedded in many different protocols and systems. No two products anywhere, even in completely different industries made by two different manufacturers in two different countries should have the same UEID (if they are not global and universal in this way, then Relying Parties receiving them will have to track other characteristics of the entity to keep entities distinct between manufacturers).

There are privacy considerations for UEIDs. See Section 8.1.

The UEID is permanent. It MUST never change for a given entity.

A UEID is constructed of a single type byte followed by the bytes that are the identifier. Several types are allowed to accommodate different industries, different manufacturing processes and to have an alternative that doesn't require paying a registration fee.

Creation of new types requires a Standards Action [RFC8126].

UEIDS are variable length to accommodate the types defined here and new types that may be defined in the future.

All implementations MUST be able to receive UEIDs up to 33 bytes long. 33 bytes is the longest defined in this document and gives necessary entropy for probabilistic uniqueness. See Appendix B.

UEIDs SHOULD NOT be longer than 33 bytes. If they are longer, there is no guarantee that a receiver will be able to accept them.

Type Byte	Type Name	Specification
0x01	RAND	This is a 128, 192 or 256-bit random number generated once and stored in the entity. This may be constructed by concatenating enough identifiers to make up an equivalent number of random bits and then feeding the concatenation through a cryptographic hash function. It may also be a cryptographic quality random number generated once at the beginning of the life of the entity and stored. It MUST NOT be smaller than 128 bits. See the length analysis in Appendix B.
0x02	IEEE EUI	This uses the IEEE company identification registry. An EUI is either an EUI-48, EUI-60 or EUI-64 and made up of an OUI, OUI-36 or a CID, different registered company identifiers, and some unique per-entity identifier. EUIs are often the same as or similar to MAC addresses. This type includes MAC-48, an obsolete name for EUI-48. (Note that while entities with multiple network interfaces may have multiple MAC addresses, there is only one UEID for an entity) [IEEE.802-2001], [OUI.Guide].
0x03	IMEI	This is a 14-digit identifier consisting of an 8-digit Type Allocation Code and a 6-digit serial number allocated by the manufacturer, which SHALL be encoded as byte string of length 14 with each byte as the digit's value (not the ASCII encoding of the digit; the digit 3 encodes as 0x03, not 0x33). The IMEI value encoded SHALL NOT include Luhn checksum or SVN information. See [ThreeGPP.IMEI].

Table 1: UEID Composition Types

UEIDs are not designed for direct use by humans (e.g., printing on the case of a device), so no textual representation is defined.

The consumer of a UEID MUST treat a UEID as a completely opaque string of bytes and not make any use of its internal structure. For example, they should not use the OUI part of a type 0x02 UEID to identify the manufacturer of the entity. Instead, they should use the "oemid" claim. See Section 4.2.3. The reasons for this are:

- UEIDs types may vary freely from one manufacturer to the next.
- New types of UEIDs may be created. For example, a type 0x07 UEID may be created based on some other manufacturer registration scheme.

- The manufacturing process for an entity is allowed to change from using one type of UEID to another. For example, a manufacturer may find they can optimize their process by switching from type 0x01 to type 0x02 or vice versa.

The type byte is needed to distinguish UEIDs of different types that by chance have the same identifier value, but do not identify the same entity. The type byte MUST be treated as part of the opaque UEID and MUST not be used to make use of the internal structure of the UEID.

A Device Identifier URN is registered for UEIDs. See Section 10.2.3.

```
$$Claims-Set-Claims //= (ueid-label => ueid-type)

ueid-type = JC<base64-url-text .size (12..44) , bstr .size (7..33)>
```

4.2.2. sueids (Semi-permanent UEIDs) Claim (SUEIDs)

The "sueids" claim conveys one or more semi-permanent UEIDs (SUEIDs). An SUEID has the same format, characteristics and requirements as a UEID, but MAY change to a different value on entity life-cycle events. An entity MAY have both a UEID and SUEIDs, neither, one or the other.

Examples of life-cycle events are change of ownership, factory reset and on-boarding into an IoT device management system. It is beyond the scope of this document to specify particular types of SUEIDs and the life-cycle events that trigger their change. An EAT profile MAY provide this specification.

There MAY be multiple SUEIDs. Each has a text string label the purpose of which is to distinguish it from others. The label MAY name the purpose, application or type of the SUEID. For example, the label for the SUEID used by FIDO Onboarding Protocol could be "FDO". It is beyond the scope of this document to specify any SUEID labeling schemes. They are use-case specific and MAY be specified in an EAT profile.

If there is only one SUEID, the claim remains a map and there still MUST be a label.

An SUEID provides functionality similar to an IEEE LDevID [IEEE.802.1AR].

There are privacy considerations for SUEIDs. See Section 8.1.

A Device Indentifier URN is registered for SUEIDs. See Section 10.2.3.

```
$$Claims-Set-Claims //= (sueids-label => sueids-type)

sueids-type = {
    + tstr => ueid-type
}
```

4.2.3. oemid (Hardware OEM Identification) Claim

The "oemid" claim identifies the Original Equipment Manufacturer (OEM) of the hardware. Any of the three forms described below MAY be used at the convenience of the claim sender. The receiver of this claim MUST be able to handle all three forms.

4.2.3.1. Random Number Based OEMID

The random number based OEMID MUST always 16 bytes (128 bits).

The OEM MAY create their own ID by using a cryptographic-quality random number generator. They would perform this only once in the life of the company to generate the single ID for said company. They would use that same ID in every entity they make. This uniquely identifies the OEM on a statistical basis and is large enough should there be ten billion companies.

The OEM MAY also use a hash function like SHA-256 and truncate the output to 128 bits. The input to the hash should be somethings that have at least 96 bits of entropy, but preferably 128 bits of entropy. The input to the hash MAY be something whose uniqueness is managed by a central registry like a domain name.

In JSON format tokens this MUST be base64url encoded.

4.2.3.2. IEEE Based OEMID

The IEEE operates a global registry for MAC addresses and company IDs. This claim uses that database to identify OEMs. The contents of the claim may be either an IEEE MA-L, MA-M, MA-S or an IEEE CID [IEEE.RA]. An MA-L, formerly known as an OUI, is a 24-bit value used as the first half of a MAC address. MA-M similarly is a 28-bit value uses as the first part of a MAC address, and MA-S, formerly known as OUI-36, a 36-bit value. Many companies already have purchased one of these. A CID is also a 24-bit value from the same space as an MA-L, but not for use as a MAC address. IEEE has published Guidelines for Use of EUI, OUI, and CID [OUI.Guide] and provides a lookup service [OUI.Lookup].

Companies that have more than one of these IDs or MAC address blocks SHOULD select one and prefer that for all their entities.

Commonly, these are expressed in Hexadecimal Representation as described in [IEEE.802-2001]. It is also called the Canonical format. When this claim is encoded the order of bytes in the bstr are the same as the order in the Hexadecimal Representation. For example, an MA-L like "AC-DE-48" would be encoded in 3 bytes with values 0xAC, 0xDE, 0x48.

This format is always 3 bytes in size in CBOR.

In JSON format tokens, this MUST be base64url encoded and always 4 bytes.

4.2.3.3. IANA Private Enterprise Number Based OEMID

IANA maintains a registry for Private Enterprise Numbers (PEN) [PEN]. A PEN is an integer that identifies an enterprise and may be used to construct an object identifier (OID) relative to the following OID arc that is managed by IANA: iso(1) identified-organization(3) dod(6) internet(1) private(4) enterprise(1).

For EAT purposes, only the integer value assigned by IANA as the PEN is relevant, not the full OID value.

In CBOR this value MUST be encoded as a major type 0 integer and is typically 3 bytes. In JSON, this value MUST be encoded as a number.

```
$$Claims-Set-Claims //= (
    oemid-label => oemid-pen / oemid-ieee / oemid-random
)

oemid-pen = int

oemid-ieee = JC<oemid-ieee-json, oemid-ieee-cbor>
oemid-ieee-cbor = bstr .size 3
oemid-ieee-json = base64-url-text .size 4

oemid-random = JC<oemid-random-json, oemid-random-cbor>
oemid-random-cbor = bstr .size 16
oemid-random-json = base64-url-text .size 24
```

4.2.4. hwmodel (Hardware Model) Claim

The "hwmodel" claim differentiates hardware models, products and variants manufactured by a particular OEM, the one identified by OEM ID in Section 4.2.3.

This claim must be unique so as to differentiate the models and products for the OEM ID. This claim does not have to be globally unique, but it can be. A receiver of this claim MUST not assume it is globally unique. To globally identify a particular product, the receiver should concatenate the OEM ID and this claim.

The granularity of the model identification is for each OEM to decide. It may be very granular, perhaps including some version information. It may be very general, perhaps only indicating top-level products.

The purpose of this claim is to identify models within protocols, not for human-readable descriptions. The format and encoding of this claim should not be human-readable to discourage use other than in protocols. If this claim is to be derived from an already-in-use human-readable identifier, it can be run through a hash function.

There is no minimum length so that an OEM with a very small number of models can use a one-byte encoding. The maximum length is 32 bytes. All receivers of this claim MUST be able to receive this maximum size.

The receiver of this claim MUST treat it as a completely opaque string of bytes, even if there is some apparent naming or structure. The OEM is free to alter the internal structure of these bytes as long as the claim continues to uniquely identify its models.

```
$$Claims-Set-Claims //= (
    hardware-model-label => hardware-model-type
)

hardware-model-type = JC<base64-url-text .size (4..44),
                         bytes .size (1..32)>
```

4.2.5. hwversion (Hardware Version) Claim

The "hwversion" claim is a text string the format of which is set by each manufacturer. The structure and sorting order of this text string can be specified using the version-scheme item from CoSWID [CoSWID]. It is useful to know how to sort versions so the newer can be distinguished from the older.

```
$$Claims-Set-Claims //=  (
    hardware-version-label => hardware-version-type
)

hardware-version-type = [
    version:  tstr,
    ? scheme:  $version-scheme
]
```

4.2.6. swname (Software Name) Claim

The "swname" claim contains a very simple free-form text value for naming the software used by the entity. Intentionally, no general rules or structure are set. This will make it unsuitable for use cases that wish precise naming.

If precise and rigourous naming of the software for the entity is needed, the "manifests" claim Section 4.2.15 may be used instead.

```
$$Claims-Set-Claims //= ( sw-name-label => tstr )
```

4.2.7. swversion (Software Version) Claim

The "swversion" claim makes use of the CoSWID version scheme data type to give a simple version for the software. A full CoSWID manifest or other type of manifest can be instead if this is too simple.

```
$$Claims-Set-Claims //= (sw-version-label => sw-version-type)

sw-version-type = [
    version:  tstr
    ? scheme:  $version-scheme
]
```

4.2.8. secboot (Secure Boot) Claim

A "secboot" claim with value of true indicates secure boot is enabled. Secure boot is considered enabled when the firmware and operating system, are under control of the manufacturer of the entity identified in the "oemid" claim described in Section 4.2.3. Control by the manufacturer of the firmware and the operating system may be by it being in ROM, being cryptographically authenticated, a combination of the two or similar.

```
$$Claims-Set-Claims //= (secure-boot-label => bool)
```

4.2.9. dbgstat (Debug Status) Claim

The "dbgstat" claim applies to entity-wide or submodule-wide debug facilities of the entity like JTAG and diagnostic hardware built into chips. It applies to any software debug facilities related to root, operating system or privileged software that allow system-wide memory inspection, tracing or modification of non-system software like user mode applications.

This characterization assumes that debug facilities can be enabled and disabled in a dynamic way or be disabled in some permanent way such that no enabling is possible. An example of dynamic enabling is one where some authentication is required to enable debugging. An example of permanent disabling is blowing a hardware fuse in a chip. The specific type of the mechanism is not taken into account. For example, it does not matter if authentication is by a global password or by per-entity public keys.

As with all claims, the absence of the "dbgstat" claim means it is not reported. A conservative interpretation might assume the enabled state.

This claim is not extensible so as to provide a common interoperable description of debug status. If a particular implementation considers this claim to be inadequate, it can define its own proprietary claim. It may consider including both this claim as a coarse indication of debug status and its own proprietary claim as a refined indication.

The higher levels of debug disabling requires that all debug disabling of the levels below it be in effect. Since the lowest level requires that all of the target's debug be currently disabled, all other levels require that too.

There is no inheritance of claims from a submodule to a superior module or vice versa. There is no assumption, requirement or guarantee that the target of a superior module encompasses the targets of submodules. Thus, every submodule must explicitly describe its own debug state. The receiver of an EAT MUST not assume that debug is turned off in a submodule because there is a claim indicating it is turned off in a superior module.

An entity may have multiple debug facilities. The use of plural in the description of the states refers to that, not to any aggregation or inheritance.

The architecture of some chips or devices may be such that a debug facility operates for the whole chip or device. If the EAT for such a chip includes submodules, then each submodule should independently report the status of the whole-chip or whole-device debug facility. This is the only way the receiver can know the debug status of the submodules since there is no inheritance.

4.2.9.1. Enabled

If any debug facility, even manufacturer hardware diagnostics, is currently enabled, then this level must be indicated.

4.2.9.2. Disabled

This level indicates all debug facilities are currently disabled. It may be possible to enable them in the future. It may also be that they were enabled in the past, but they are currently disabled.

4.2.9.3. Disabled Since Boot

This level indicates all debug facilities are currently disabled and have been so since the entity booted/started.

4.2.9.4. Disabled Permanently

This level indicates all non-manufacturer facilities are permanently disabled such that no end user or developer can enable them. Only the manufacturer indicated in the "oemid" claim can enable them. This also indicates that all debug facilities are currently disabled and have been so since boot/start.

4.2.9.5. Disabled Fully and Permanently

This level indicates that all debug facilities for the entity are permanently disabled.

```
$$Claims-Set-Claims //= ( debug-status-label => debug-status-type )

debug-status-type = ds-enabled /
                    disabled /
                    disabled-since-boot /
                    disabled-permanently /
                    disabled-fully-and-permanently

ds-enabled                     = JC< "enabled", 0 >
disabled                       = JC< "disabled", 1 >
disabled-since-boot            = JC< "disabled-since-boot", 2 >
disabled-permanently           = JC< "disabled-permanently", 3 >
disabled-fully-and-permanently = JC< "disabled-fully-and-permanently",
                                     4 >
```

4.2.10. location (Location) Claim

The "location" claim gives the location of the entity from which the attestation originates. It is derived from the W3C Geolocation API [W3C.GeoLoc]. The latitude, longitude, altitude and accuracy must conform to [WGS84]. The altitude is in meters above the [WGS84] ellipsoid. The

two accuracy values are positive numbers in meters. The heading is in degrees relative to true north. If the entity is stationary, the heading is NaN (floating-point not-a-number). The speed is the horizontal component of the entity velocity in meters per second.

The location may have been cached for a period of time before token creation. For example, it might have been minutes or hours or more since the last contact with a GPS satellite. Either the timestamp or age data item can be used to quantify the cached period. The timestamp data item is preferred as it a non-relative time.

The age data item can be used when the entity doesn't know what time it is either because it doesn't have a clock or it isn't set. The entity MUST still have a "ticker" that can measure a time interval. The age is the interval between acquisition of the location data and token creation.

See location-related privacy considerations in Section 8.2.

```
$$Claims-Set-Claims //= (location-label => location-type)

location-type = {
    latitude => number,
    longitude => number,
    ? altitude => number,
    ? accuracy => number,
    ? altitude-accuracy => number,
    ? heading => number,
    ? speed => number,
    ? timestamp => ~time-int,
    ? age => uint
}

latitude          = JC< "latitude",          1 >
longitude         = JC< "longitude",         2 >
altitude          = JC< "altitude",          3 >
accuracy          = JC< "accuracy",          4 >
altitude-accuracy = JC< "altitude-accuracy", 5 >
heading           = JC< "heading",           6 >
speed             = JC< "speed",             7 >
timestamp         = JC< "timestamp",         8 >
age               = JC< "age",               9 >
```

4.2.11. uptime (Uptime) Claim

The "uptime" claim MUST contain a value that represents the number of seconds that have elapsed since the entity or submod was last booted.

```
$$Claims-Set-Claims //= (uptime-label => uint)
```

4.2.12. bootcount (Boot Count) Claim

The "bootcount" claim contains a count of the number times the entity or submod has been booted. Support for this claim requires a persistent storage on the device.

```
$$Claims-Set-Claims //= (boot-count-label => uint)
```

4.2.13. bootseed (Boot Seed) Claim

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 (e.g., a certain UEID).

This value is usually public. It is not a secret and MUST NOT be used for any purpose that a secret seed is needed, such as seeding a random number generator.

There are privacy considerations for Boot Seed. See Section 8.3.

```
$$Claims-Set-Claims //=  (boot-seed-label => binary-data)
```

4.2.14. dloas (Digital Letters of Approval) Claim

The "dloas" claim conveys one or more Digital Letters of Approval (DLOAs)). A DLOA [DLOA] is a document that describes a certification that an entity has received. Examples of certifications represented by a DLOA include those issued by Global Platform and those based on Common Criteria. The DLOA is unspecific to any particular certification type or those issued by any particular organization.

This claim is typically issued by a verifier, not an attester. Verifiers MUST NOT issue this claim unless the entity has received the certification indicated by the DLOA.

This claim MAY contain more than one DLOA. If multiple DLOAs are present, verifiers MUST NOT issue this claim unless the entity has received all of the certifications.

DLOA documents are always fetched from a registrar that stores them. This claim contains several data items used to construct a URL for fetching the DLOA from the particular registrar.

This claim MUST be encoded as an array with either two or three elements. The first element MUST be the URI for the registrar. The second element MUST be a platform label indicating which platform was certified. If the DLOA applies to an application, then the third element is added which MUST be an application label. The method of constructing the registrar URI, platform label and possibly application label is specified in [DLOA].

```
$$Claims-Set-Claims //= (
    dloas-label => [ + dloa-type ]
)

dloa-type = [
    dloa_registrar: general-uri
    dloa_platform_label: text
    ? dloa_application_label: text
]
```

4.2.15. manifests (Software Manifests) Claim

The "manifests" claim contains descriptions of software present on the entity. These manifests are installed on the entity when the software is installed or are created as part of the installation process. Installation is anything that adds software to the entity, possibly factory installation, the user installing elective applications and so on. The defining characteristic is they are created by the software manufacturer. The purpose of these claims in an EAT is to relay them without modification to the verifier and possibly to the relying party.

Some manifests may be signed by their software manufacturer before they are put into this EAT claim. When such manifests are put into this claim, the manufacturer's signature SHOULD be included. For example, the manifest might be a CoSWID signed by the software manufacturer, in which case the full signed CoSWID should be put in this claim.

This claim allows multiple formats for the manifest. For example, the manifest may be a CBOR-format CoSWID, an XML-format SWID or other. Identification of the type of manifest is always by a CoAP Content-Format integer [RFC7252]. If there is no CoAP identifier registered for the manifest format, one should be registered, perhaps in the experimental or first-come-first-served range.

This claim MUST be an array of one or more manifests. Each manifest in the claim MUST be an array of two. The first item in the array of two MUST be an integer CoAP Content-Format identifier. The second item is MUST be the actual manifest.

In JSON-format tokens the manifest, whatever format it is, MUST be placed in a text string. When a non-text format manifest like a CBOR-encoded CoSWID is put in a JSON-encoded token, the manifest MUST be base-64 encoded.

This claim allows for multiple manifests in one token since multiple software packages are likely to be present. The multiple manifests MAY be of different formats. In some cases EAT submodules may be used instead of the array structure in this claim for multiple manifests.

When the [CoSWID] format is used, it MUST be a payload CoSWID, not an evidence CoSWID.

A [SUIT.Manifest] may be used as a manifest.

This document registers CoAP Content Formats for CycloneDX [CycloneDX] and SPDX [SPDX] so they can be used as a manifest.

This claim is extensible for use of manifest formats beyond those mentioned in this document. No particular manifest format is preferred. For manifest interoperability, an EAT profile, Section 6, should be used that specifies what manifest format(s) are allowed.

```
$$Claims-Set-Claims //= (
    manifests-label => manifests-type
)

manifests-type = [+ manifest-format]

manifest-format = [
    content-type:   coap-content-format,
    content-format: JC< $manifest-body-json,
                        $manifest-body-cbor >
]

$manifest-body-cbor /= bytes .cbor untagged-coswid
$manifest-body-json /= base64-url-text

$manifest-body-cbor /= bytes .cbor SUIT_Envelope
$manifest-body-json /= base64-url-text

$manifest-body-cbor /= spdx-json
$manifest-body-json /= spdx-json

spdx-json = text

$manifest-body-cbor /= cyclone-dx-json
$manifest-body-cbor /= cyclone-dx-xml
$manifest-body-json /= cyclone-dx-json
$manifest-body-json /= cyclone-dx-xml
cyclone-dx-json = text
cyclone-dx-xml  = text

suit-directive-process-dependency = 19
```

4.2.16. measurements (Measurements) Claim

The "measurements" claim contains descriptions, lists, evidence or measurements of the software that exists on the entity or any other measurable subsystem of the entity (e.g. hash of sections of a file system or non-volatile memory). The defining characteristic of this claim is that its contents are created by processes on the entity that inventory, measure or otherwise characterize the software on the entity. The contents of this claim do not originate from the manufacturer of the measurable subsystem (e.g. developer of a software library).

This claim can be a [CoSWID]. When the CoSWID format is used, it MUST be evidence CoSWIDs, not payload CoSWIDS.

Formats other than CoSWID can be used. The identification of format is by CoAP Content Format, the same as the "manifests" claim in Section 4.2.15.

```
$$Claims-Set-Claims //= (
    measurements-label => measurements-type
)

measurements-type = [+ measurements-format]

measurements-format = [
    content-type:   coap-content-format,
    content-format: JC< $$measurements-body-json,
                        $$measurements-body-cbor >
]

$$measurements-body-cbor /= bytes .cbor untagged-coswid
$$measurements-body-json /= base64-url-text
```

4.2.17. measres (Software Measurement Results) Claim

The "measres" claim is a general-purpose structure for reporting comparison of measurements to expected reference values. This claim provides a simple standard way to report the result of a comparison as success, failure, fail to run, ...

It is the nature of measurement systems that they are specific to the operating system, software and hardware of the entity that is being measured. It is not possible to standardize what is measured and how it is measured across platforms, OS's, software and hardware. The recipient must obtain the information about what was measured and what it indicates for the characterization of the security of the entity from the provider of the measurement system. What this claim provides is a standard way to report basic success or failure of the measurement. In some use cases it is valuable to know if measurements succeeded or failed in a general way even if the details of what was measured is not characterized.

This claim MAY be generated by the verifier and sent to the relying party. For example, it could be the results of the verifier comparing the contents of the "measurements" claim, Section 4.2.16, to reference values.

This claim MAY also be generated on the entity if the entity has the ability for one subsystem to measure and evaluate another subsystem. For example, a TEE might have the ability to measure the software of the rich OS and may have the reference values for the rich OS.

Within an entity, attestation target or submodule, multiple results can be reported. For example, it may be desirable to report the results for measurements of the file system, chip configuration, installed software, running software and so on.

Note that this claim is not for reporting the overall result of a verifier. It is solely for reporting the result of comparison to reference values.

An individual measurement result is an array of two, an identifier of the measurement and an enumerated type that is the result. The range and values of the measurement identifier varies from one measurement scheme to another.

Each individual measurement result is part of a group that may contain many individual results. Each group has a text string that names it, typically the name of the measurement scheme or system.

The claim itself consists of one or more groups.

The values for the results enumerated type are as follows:

1 - comparison successful Indicates successful comparison to reference values.

2 - comparison fail The comparison was completed and did not compare correctly to the reference values.

3 - comparison not run The comparison was not run. This includes error conditions such as running out of memory.

4 - measurement absent The particular measurement was not available for comparison.

```
$$Claims-Set-Claims //= (
    measurement-results-label =>
        [ + measurement-results-group ] )

measurement-results-group = [
    measurement-system: tstr,
    measruement-results: [ + individual-result ]
]

individual-result = [
    results-id: tstr / binary-data,
    result:     result-type,
]

result-type = comparison-successful /
              comparison-fail /
              comparison-not-run /
              measurement-absent

comparison-successful    = JC< "success",       1 >
comparison-fail          = JC< "fail",          2 >
comparison-not-run       = JC< "not-run",       3 >
measurement-absent       = JC< "absent",        4 >
```

4.2.18. submods (Submodules)

Some devices are complex, having many subsystems. A mobile phone is a good example. It may have several connectivity subsystems for communications (e.g., Wi-Fi and cellular). It may have subsystems for low-power audio and video playback. It may have multiple security-oriented subsystems like a TEE and a Secure Element.

The claims for a subsystem can be grouped together in a submodule or submod.

The submods are in a single map/object, one entry per submodule. There is only one submods map/object in a token. It is identified by its specific label. It is a peer to other claims, but it is not called a claim because it is a container for a claims set rather than an individual claim. This submods part of a token allows what might be called recursion. It allows claims sets inside of claims sets inside of claims sets...

4.2.18.1. Submodule Types

The following sections define the three types of submodules:

- A submodule Claims-Set
- A nested token, which can be any valid EAT token, CBOR or JSON
- The digest of a detached Claims-Set

```
$$Claims-Set-Claims //= (submods-label => { + text => Submodule })

Submodule = Claims-Set / Nested-Token / Detached-Submodule-Digest
```

4.2.18.1.1. Submodule Claims-Set

This is a subordinate Claims-Set containing claims about a submodule, a subordinate entity.

The submodule Claims-Set is produced by the same attester as the surrounding token. It is secured by the same mechanism as the enclosing token (e.g., it is signed by the same attestation key). It roughly corresponds to an attesting environment, as described in the RATS architecture.

It may contain claims that are the same as its surrounding token or superior submodules. For example, the top-level of the token may have a UEID, a submod may have a different UEID and a further subordinate submodule may also have a UEID.

The encoding of a submodule Claims-Set MUST be the same as the encoding as the token it is part of.

The data type for this type of submodule is a map/object. It is identified when decoding by its type being a map/object.

4.2.18.1.2. Nested Token

This type of submodule is a fully formed complete token. It is typically produced by a separate attester. It is typically used by a composite device as described in RATS Architecture [RATS.Architecture] In being a submodule of the surrounding token, it is cryptographically bound to the surrounding token. If it was conveyed in parallel with the surrounding token, there would be no such binding and attackers could substitute a good attestation from another device for the attestation of an errant subsystem.

A nested token does not need to use the same encoding as the enclosing token. This is to allow composite devices to be built without regards to the encoding supported by their attesters. Thus, a CBOR-encoded token like a CWT can have a JWT as a nested token submodule and vice versa.

4.2.18.1.2.1. Surrounding EAT is CBOR-Encoded

This describes the encoding and decoding of CBOR or JSON-encoded tokens nested inside a CBOR-encoded token.

If the nested token is CBOR-encoded, then it MUST be a CBOR tag and MUST be wrapped in a byte string. The tag identifies whether the nested token is a CWT, a CBOR-encoded detached EAT bundle, or some other CBOR-format token defined in the future. A nested CBOR-encoded token that is not a CBOR tag is NOT allowed.

If the nested token is JSON-encoded, then the data item MUST be a text string containing JSON. The JSON is defined in CDDL by JSON-Nested-Token in the next section.

When decoding, if a byte string is encountered, it is known to be a nested CBOR-encoded token. The byte string wrapping is removed. The type of the token is determined by the CBOR tag.

When decoding, if a text string is encountered, it is known to be a JSON-encoded token. The two-item array is decoded and tells the type of the JSON-encoded token.

```
Nested-Token = CBOR-Nested-Token

CBOR-Nested-Token =
    JSON-Token-Inside-CBOR-Token /
    CBOR-Token-Inside-CBOR-Token

CBOR-Token-Inside-CBOR-Token = bstr .cbor $$EAT-CBOR-Tagged-Token

JSON-Token-Inside-CBOR-Token = tstr
```

4.2.18.1.2.2. Surrounding EAT is JSON-Encoded

This describes the encoding and decoding of CBOR or JSON-encoded tokens nested inside a JSON-encoded token.

The nested token MUST be an array of two, a text string type indicator and the actual token.

The string identifying the JSON-encoded token MUST be one of the following:

"JWT": The second array item MUST be a JWT formatted according to [RFC7519]

"CBOR": The second array item must be some base64url-encoded CBOR that is a tag, typically a CWT or CBOR-encoded detached EAT bundle

"BUNDLE": The second array item MUST be a JSON-encoded detached EAT bundle as defined in this document.

Additional types may be defined by a standards action.

When decoding, the array of two is decoded. The first item indicates the type and encoding of the nested token. If the type string is not "CBOR", then the token is JSON-encoded and of the type indicated by the string.

If the type string is "CBOR", then the token is CBOR-encoded. The base64url encoding is removed. The CBOR-encoded data is then decoded. The type of nested token is determined by the CBOR-tag. It is an error if the CBOR is not a tag.

```
Nested-Token = JSON-Nested-Token

JSON-Nested-Token = [
   type : "JWT" / "CBOR" / "BUNDLE",
   nested-token : JWT-Message /
                  CBOR-Token-Inside-JSON-Token /
                  Detached-EAT-Bundle
]

CBOR-Token-Inside-JSON-Token = base64-url-text
```

4.2.18.1.3. Detached Submodule Digest

This is type of submodule equivalent to a Claims-Set submodule, except the Claims-Set is conveyed separately outside of the token.

This type of submodule consists of a digest made using a cryptographic hash of a Claims-Set. The Claims-Set is not included in the token. It is conveyed to the verifier outside of the token. The submodule containing the digest is called a detached digest. The separately conveyed Claims-Set is called a detached claims set. A detached Claims-Set can include other submodules including nested tokens and detached digests.

The input to the digest algorithm is directly the CBOR or JSON-encoded Claims-Set of the submodule. There is no byte-string wrapping or base 64 encoding.

The encoding type of the detached claims set is part of the carrying protocol and varies from protocol to protocol. For example, a detached EAT bundle uses mechanisms defined in this document. Other use cases may use a content/media type.

The primary use for this is to facilitate the implementation of a small and secure attester, perhaps purely in hardware. This small, secure attester implements COSE signing and only a few claims, perhaps just UEID and hardware identification. It has inputs for digests of submodules, perhaps 32-byte hardware registers. Software running on the device constructs larger claim sets, perhaps very large, encodes them and digests them. The digests are written into the small secure attesters registers. The EAT produced by the small secure attester only contains the UEID, hardware identification and digests and is thus simple enough to be implemented in hardware. Probably, every data item in it is of fixed length.

The data type for this type of submodule MUST be an array It contains two data items, a hash algorithm identifier and a byte string containing the digest.

The hash algorithm identifier is always from the COSE Algorithm registry, [IANA.COSE.Algorithms]. Either the integer or string identifier may be used. The hash algorithm identifier is never from the JOSE Algorithm registry.

When decoding a CBOR format token, the detached digest type is distinguished from the other types by it being an array. In CBOR encoded tokens none of other submodule types are arrays.

When decoding a JSON format token, a little more work is required because both the nested token and detached digest types are an array. To distinguish the nested token from the detached digest, the first element in the array is examined. If it is "JWT" or "BUNDLE", then the submodule is a nested token. Otherwise it will contain an algorithm identifier and is a detached digest.

A detached EAT bundle, described in Section 5, may be used to convey detached claims sets and the token with their detached digests. EAT, however, doesn't require use of a detached EAT bundle. Any other protocols may be used to convey detached claims sets and the token with their detached digests. Note that since detached Claims-Sets are signed, protocols conveying them must make sure they are not modified in transit.

```
Detached-Submodule-Digest = [
   hash-algorithm : text / int,
   digest         : binary-data
]
```

4.2.18.2. No Inheritance

The subordinate modules do not inherit anything from the containing token. The subordinate modules must explicitly include all of their claims. This is the case even for claims like an EAT nonce (Section 4.1).

This rule is in place for simplicity. It avoids complex inheritance rules that might vary from one type of claim to another.

4.2.18.3. Security Levels

The security level of the non-token subordinate modules should always be less than or equal to that of the containing modules in the case of non-token submodules. It makes no sense for a module of lesser security to be signing claims of a module of higher security. An example of this is a TEE signing claims made by the non-TEE parts (e.g. the high-level OS) of the device.

The opposite may be true for the nested tokens. They usually have their own more secure key material. An example of this is an embedded secure element.

4.2.18.4. Submodule Names

The label or name for each submodule in the submods map is a text string naming the submodule. No submodules may have the same name.

4.3. Claims Describing the Token

The claims in this section provide meta data about the token they occur in. They do not describe the entity.

They may appear in evidence or attestation results. When these claims appear in evidence, they SHOULD not be passed through the verifier into attestation results.

4.3.1. iat (Timestamp) Claim

The "iat" claim defined in CWT and JWT is used to indicate the date-of-creation of the token, the time at which the claims are collected and the token is composed and signed.

The data for some claims may be held or cached for some period of time before the token is created. This period may be long, even days. Examples are measurements taken at boot or a geographic position fix taken the last time a satellite signal was received. There are individual timestamps associated with these claims to indicate their age is older than the "iat" timestamp.

CWT allows the use floating-point for this claim. EAT disallows the use of floating-point. An EAT token MUST NOT contain an "iat" claim in floating-point format. Any recipient of a token with a floating-point format "iat" claim MUST consider it an error.

A 64-bit integer representation of the CBOR epoch-based time [RFC8949] used by this claim can represent a range of +/- 500 billion years, so the only point of a floating-point timestamp is to have precession greater than one second. This is not needed for EAT.

4.3.2. eat_profile (EAT Profile) Claim

See Section 6 for the detailed description of an EAT profile.

The "eat_profile" claim identifies an EAT profile by either a URL or an OID. Typically, the URI will reference a document describing the profile. An OID is just a unique identifier for the profile. It may exist anywhere in the OID tree. There is no requirement that the named document be publicly accessible. The primary purpose of the "eat_profile" claim is to uniquely identify the profile even if it is a private profile.

The OID is always absolute and never relative.

See Section 7.2.1 for OID and URI encoding.

```
$$Claims-Set-Claims //= (profile-label => general-uri / general-oid)
```

4.3.3. intuse (Intended Use) Claim

EAT's may be used in the context of several different applications. The "intuse" claim provides an indication to an EAT consumer about the intended usage of the token. This claim can be used as a way for an application using EAT to internally distinguish between different ways it uses EAT.

1 - Generic: Generic attestation describes an application where the EAT consumer requires the most up-to-date security assessment of the attesting entity. It is expected that this is the most commonly-used application of EAT.

2- Registration: Entities that are registering for a new service may be expected to provide an attestation as part of the registration process. This "intuse" setting indicates that the attestation is not intended for any use but registration.

3 - Provisioning: Entities may be provisioned with different values or settings by an EAT consumer. Examples include key material or device management trees. The consumer may require an EAT to assess entity security state of the entity prior to provisioning.

4 - Certificate Issuance Certification Authorities (CA's) may require attestations prior to the issuance of certificates related to keypairs hosted at the entity. An EAT may be used as part of the certificate signing request (CSR).

5 - Proof-of-Possession: An EAT consumer may require an attestation as part of an accompanying proof-of-possession (PoP) application. More precisely, a PoP transaction is intended to provide to the recipient cryptographically-verifiable proof that the sender has possession of a key. This kind of attestation may be necceesary to verify the security state of the entity storing the private key used in a PoP application.

```
$$Claims-Set-Claims //= ( intended-use-label => intended-use-type )

intended-use-type = generic /
                    registration /
                    provisioning /
                    csr /
                    pop

generic      = JC< "generic",      1 >
registration = JC< "registration", 2 >
provisioning = JC< "provisioning", 3 >
csr          = JC< "csr",          4 >
pop          = JC< "pop",          5 >
```

5. Detached EAT Bundles

A detached EAT bundle is a structure to convey a fully-formed and signed token plus detached claims set that relate to that token. It is a top-level EAT message like a CWT or JWT. It can be occur any place that CWT or JWT messages occur. It may also be sent as a submodule.

A detached EAT bundle has two main parts.

The first part is a full top-level token. This top-level token must have at least one submodule that is a detached digest. This top-level token may be either CBOR or JSON-encoded. It may be a CWT, or JWT but not a detached EAT bundle. It may also be some future-defined token type. The same mechanism for distinguishing the type for nested token submodules is used here.

The second part is a map/object containing the detached Claims-Sets corresponding to the detached digests in the full token. When the detached EAT bundle is CBOR-encoded, each Claims-Set is wrapped in a byte string. When the detached EAT bundle is JSON-encoded, each Claims-Set is base64url encoded. All the detached Claims-Sets MUST be encoded in the same format as the detached EAT bundle. No mixing of encoding formats is allowed for the Claims-Sets in a detached EAT bundle.

For CBOR-encoded detached EAT bundles, tag TBD602 can be used to identify it. The normal rules apply for use or non-use of a tag. When it is sent as a submodule, it is always sent as a tag to distinguish it from the other types of nested tokens.

The digests of the detached claims sets are associated with detached Claims-Sets by label/name. It is up to the constructor of the detached EAT bundle to ensure the names uniquely identify the detachedclaims sets. Since the names are used only in the detached EAT bundle, they can be very short, perhaps one byte.

```
BUNDLE-Messages = BUNDLE-Tagged-Message / BUNDLE-Untagged-Message

BUNDLE-Tagged-Message   = #6.TBD(BUNDLE-Untagged-Message)
BUNDLE-Untagged-Message = Detached-EAT-Bundle

Detached-EAT-Bundle = [
    main-token : Nested-Token,
    detached-claims-sets: {
        + tstr => JC<json-wrapped-claims-set,
                     cbor-wrapped-claims-set>
    }
]

json-wrapped-claims-set = base64-url-text

cbor-wrapped-claims-set = bstr .cbor Claims-Set
```

6. Profiles

EAT makes normative use of CBOR, JSON, COSE, JOSE, CWT and JWT. Most of these have implementation options to accommodate a range of use cases.

For example, COSE doesn't require a particular set of cryptographic algorithms so as to accommodate different usage scenarios and evolution of algorithms over time. Section 10 of [RFC9052] describes the profiling considerations for COSE.

The use of encryption is optional for both CWT and JWT. Section 8 of [RFC7519] describes implementation requirement and recommendations for JWT.

Similarly, CBOR provides indefinite length encoding which is not commonly used, but valuable for very constrained devices. For EAT itself, in a particular use case some claims will be used and others will not. Section 4 of [RFC8949] describes serialization considerations for CBOR.

For example a mobile phone use case may require the device make and model, and prohibit UEID and location for privacy policy. The general EAT standard retains all this flexibility because it too is aimed to accommodate a broad range of use cases.

It is necessary to explicitly narrow these implementation options to guarantee interoperability. EAT chooses one general and explicit mechanism, the profile, to indicate the choices made for these implementation options for all aspects of the token.

Below is a list of the various issues that should be addressed by a profile.

The "eat_profile" claim in Section 4.3.2 provides a unique identifier for the profile a particular token uses.

A profile can apply to evidence or to attestation results or both.

6.1. Format of a Profile Document

A profile document doesn't have to be in any particular format. It may be simple text, something more formal or a combination.

A profile may define, and possibly register, one or more new claims if needed. A profile may also reuse one or more already defined claims, either as-is or with values constrained to a subset or subrange.

6.2. List of Profile Issues

The following is a list of EAT, CWT, JWT, COSE, JOSE and CBOR options that a profile should address.

6.2.1. Use of JSON, CBOR or both

A profile should specify whether CBOR, JSON or both may be sent. A profile should specify that the receiver can accept all encoding formats that the sender is allowed to send.

This should be specified for the top-level and all nested tokens. For example, a profile might require all nested tokens to be of the same encoding of the top level token.

6.2.2. CBOR Map and Array Encoding

A profile should specify whether definite-length arrays/maps, indefinite-length arrays/maps or both may be sent. A profile should specify that the receiver be able to accept all length encodings that the sender is allowed to send.

This applies to individual EAT claims, CWT and COSE parts of the implementation.

For most use cases, specifying that only definite-length arrays/maps may be sent is suitable.

6.2.3. CBOR String Encoding

A profile should specify whether definite-length strings, indefinite-length strings or both may be sent. A profile should specify that the receiver be able to accept all types of string encodings that the sender is allowed to send.

For most use cases, specifying that only definite-length strings may be sent is suitable.

6.2.4. CBOR Preferred Serialization

A profile should specify whether or not CBOR preferred serialization must be sent or not. A profile should specify the receiver be able to accept preferred and/or non-preferred serialization so it will be able to accept anything sent by the sender.

6.2.5. CBOR Tags

The profile should specify whether the token should be a CWT Tag or not.

When COSE protection is used, the profile should specify whether COSE tags are used or not. Note that RFC 8392 requires COSE tags be used in a CWT tag.

Often a tag is unnecessary because the surrounding or carrying protocol identifies the object as an EAT.

6.2.6. COSE/JOSE Protection

COSE and JOSE have several options for signed, MACed and encrypted messages. JWT may use the JOSE NULL protection option. It is possible to implement no protection, sign only, MAC only, sign then encrypt and so on. All combinations allowed by COSE, JOSE, JWT, and CWT are allowed by EAT.

A profile should specify all signing, encryption and MAC message formats that may be sent. For example, a profile might allow only COSE_Sign1 to be sent. For another example, a profile might allow COSE_Sign and COSE_Encrypt to be sent to carry multiple signatures for post quantum cryptography and to use encryption to provide confidentiality.

A profile should specify the receiver accepts all message formats that are allowed to be sent.

When both signing and encryption are allowed, a profile should specify which is applied first.

6.2.7. COSE/JOSE Algorithms

See the section on "Application Profiling Considerations" in [RFC9052] for a discussion on selection of cryptgraphic algorithms and related issues.

The profile document should list the COSE algorithms that a verifier must implement. The attester will select one of them. Since there is no negotiation, the verifier should implement all algorithms listed in the profile. If detached submodule digests are used, the COSE algorithms allowed for their digests should also be in the profile.

6.2.8. Detached EAT Bundle Support

A profile should specify whether or not a detached EAT bundle (Section 5) can be sent. A profile should specify that a receiver be able to accept a detached EAT bundle if the sender is allowed to send it.

6.2.9. Key Identification

A profile should specify what must be sent to identify the verification, decryption or MAC key or keys. If multiple methods of key identification may be sent, a profile should require the receiver support them all.

Appendix F describes a number of methods for identifying verification keys. When encryption is used, there are further considerations. In some cases key identification may be very simple and in others involve a multiple components. For example, it may be simple through use of COSE key ID or it may be complex through use of an X.509 certificate hierarchy.

While not always possible, a profile should specify, or make reference to, a full end-end specification for key identification. For example, a profile should specify in full detail how COSE key IDs are to be created, their lifecycle and such rather than just specifying that a COSE key ID be used. For example, a profile should specify the full details of an X.509 hierarchy including extension processing, algorithms allowed and so on rather than just saying X.509 certificate are used. Though not always possible, ideally, a profile should be a complete specification for key identification for both the sender and the receiver such that interoperability is guaranteed.

6.2.10. Endorsement Identification

Similar to, or perhaps the same as verification key identification, the profile may wish to specify how endorsements are to be identified. However note that endorsement identification is optional, where as key identification is not.

6.2.11. Freshness

Security considerations Section 9.3 requires a mechanism to provide freshness. This may be the EAT nonce claim in Section 4.1, or some claim or mechanism defined outside this document. The section on freshness in [RATS.Architecture] describes several options. A profile should specify which freshness mechanism or mechanisms can be used.

If the EAT nonce claim is used, a profile should specify whether multiple nonces may be sent. If a profile allows multiple nonces to be sent, it should require the receiver to process multiple nonces.

6.2.12. Claims Requirements

A profile may define new claims that are not defined in this document.

This document requires an EAT receiver must accept all claims it does not understand. A profile for a specific use case may reverse this and allow a receiver to reject tokens with claims it does not understand. A profile for a specific use case may specify that specific claims are prohibited.

A profile for a specific use case may modify this and specify that some claims are required.

A profile may constrain the definition of claims that are defined in this document or elsewhere. For example, a profile may require the EAT nonce be a certain length or the "location" claim always include the altitude.

Some claims are "pluggable" in that they allow different formats for their content. The "manifests" claim (Section 4.2.15) along with the measurement and "measurements" (Section 4.2.16)) claims are examples of this, allowing the use of CoSWID, TEEP Manifests and other formats. A profile should specify which formats are allowed to be sent, with the assumption that the corresponding COAP content types have been registered. A profile should require the receiver to accept all formats that are allowed to be sent.

Further, if there is variation within a format that is allowed, the profile should specify which variations can be sent. For example, there are variations in the CoSWID format. A profile that require the receiver to accept all variations that are allowed to be sent.

6.3. The Constrained Device Standard Profile

It is anticipated that there will be many profiles defined for EAT for many different use cases. This section standardizes one profile that is good for many constrained device use cases.

The identifier for this profile is "https://www.rfc-editor.org/rfc/rfcTBD".

Issue	Profile Definition
CBOR/JSON	CBOR only
CBOR Encoding	Only definite length maps and arrays are allowed
CBOR Encoding	Only definite length strings are allowed
CBOR Serialization	Only preferred serialization is allowed
COSE Protection	Only COSE_Sign1 format is used
Algorithms	Receiver MUST accept ES256, ES384 and ES512; sender MUST send one of these
Detached EAT Bundle Usage	Detached EAT bundles may not be sent with this profile
Verification Key Identification	Either the COSE kid or the UEID MUST be used to identify the verication key. If both are present, the kid takes precedence
Endorsements	This profile contains no endorsement identifier
Nonce	A new single unique nonce MUST be used for every token request
Claims	No requirement is made on the presence or absence of claims other than requiring an EAT nonce. As per general EAT rules, the receiver MUST not error out on claims it doesn't understand.

Table 2

Strictly speaking, slight modifications such use of a different means of key identification are a divergence from this profile and MUST use a different profile identifier.

A profile that is similar to this can be defined and/or standardized by making normative reference to this and adding other requirements. Such a definition MUST have a different profile identifier.

7. Encoding and Collected CDDL

An EAT is fundamentally defined using CDDL. This document specifies how to encode the CDDL in CBOR or JSON. Since CBOR can express some things that JSON can't (e.g., tags) or that are expressed differently (e.g., labels) there is some CDDL that is specific to the encoding format.

7.1. Claims-Set and CDDL for CWT and JWT

CDDL was not used to define CWT or JWT. It was not available at the time.

This document defines CDDL for both CWT and JWT. This document does not change the encoding or semantics of anything in a CWT or JWT.

A Claims-Set is the central data structure for EAT, CWT and JWT. It holds all the claims and is the structure that is secured by signing or other means. It is not possible to define EAT, CWT, or JWT in CDDL without it. The CDDL definition of Claims-Set here is applicable to EAT, CWT and JWT.

This document specifies how to encode a Claims-Set in CBOR or JSON.

With the exception of nested tokens and some other externally defined structures (e.g., SWIDs) an entire Claims-Set must be in encoded in either CBOR or JSON, never a mixture.

CDDL for the seven claims defined by [RFC8392] and [RFC7519] is included here.

7.2. Encoding Data Types

This makes use of the types defined in [RFC8610] Appendix D, Standard Prelude.

7.2.1. Common Data Types

time-int is identical to the epoch-based time, but disallows floating-point representation.

The OID encoding from [RFC9090] is used without the tag number in CBOR-encoded tokens. In JSON tokens OIDs are a text string in the common form of "nn.nn.nn...".

Unless expliclity indicated, URIs are not the URI tag defined in [RFC8949]. They are just text strings that contain a URI.

```
time-int = #6.1(int)

binary-data = JC< base64-url-text, bstr>

base64-url-text = tstr .regexp "[A-Za-z0-9_=-]+"

general-oid = JC< json-oid, ~oid >

json-oid = tstr .regexp "([0-2])((\.0)|(\.[1-9][0-9]*))*"

general-uri = JC< text, ~uri >

coap-content-format = uint .le 65535
```

7.2.2. JSON Interoperability

JSON should be encoded per [RFC8610] Appendix E. In addition, the following CDDL types are encoded in JSON as follows:

- bstr - must be base64url encoded
- time - must be encoded as NumericDate as described section 2 of [RFC7519].
- string-or-uri - must be encoded as StringOrURI as described section 2 of [RFC7519].
- uri - must be a URI [RFC3986].
- oid - encoded as a string using the well established dotted-decimal notation (e.g., the text "1.2.250.1").

The CDDL generic "JC< >" is used in most places where there is a variance between CBOR and JSON. The first argument is the CDDL for JSON and the second is CDDL for CBOR.

7.2.3. Labels

Most map labels, Claims-Keys, Claim-Names and enumerated-type values are integers for CBOR-encoded tokens and strings for JSON-encoded tokens. When this is the case the "JC < >" CDDL construct is used to give both the integer and string values.

7.2.4. CBOR Interoperability

CBOR allows data items to be serialized in more than one form to accommodate a variety of use cases. This is addressed in Section 6.

7.3. Collected CDDL

7.3.1. Payload CDDL

This CDDL defines all the EAT Claims that are added to the main definition of a Claim-Set in Appendix D. Claims-Set is the payload for CWT, JWT and potentially other token types. This is for both CBOR and JSON. When there is variation between CBOR and JSON, the JC<> CDDL generic defined in Appendix D.

This CDDL uses, but doesn't define Nested-Token because its definition varies between CBOR and JSON and the JC<> generic can't be used to define it. Nested-Token is the one place that that a CBOR token can be nested inside a JSON token and vice versa. Nested-Token is defined in the following sections.

```
time-int = #6.1(int)

binary-data = JC< base64-url-text, bstr>

base64-url-text = tstr .regexp "[A-Za-z0-9_=-]+"

general-oid = JC< json-oid, ~oid >

json-oid = tstr .regexp "([0-2])((\.0)|(\.[1-9][0-9]*))*"

general-uri = JC< text, ~uri >

coap-content-format = uint .le 65535


$$Claims-Set-Claims //=
    (nonce-label => nonce-type / [ 2* nonce-type ])

nonce-type = JC< tstr .size (10..74), bstr .size (8..64)>


$$Claims-Set-Claims //= (ueid-label => ueid-type)

ueid-type = JC<base64-url-text .size (12..44) , bstr .size (7..33)>

$$Claims-Set-Claims //= (sueids-label => sueids-type)

sueids-type = {
    + tstr => ueid-type
}

$$Claims-Set-Claims //= (
    oemid-label => oemid-pen / oemid-ieee / oemid-random
)

oemid-pen = int

oemid-ieee = JC<oemid-ieee-json, oemid-ieee-cbor>
oemid-ieee-cbor = bstr .size 3
oemid-ieee-json = base64-url-text .size 4

oemid-random = JC<oemid-random-json, oemid-random-cbor>
oemid-random-cbor = bstr .size 16
oemid-random-json = base64-url-text .size 24


$$Claims-Set-Claims //=  (
    hardware-version-label => hardware-version-type
)

hardware-version-type = [
    version:  tstr,
    ? scheme:  $version-scheme
]

$$Claims-Set-Claims //= (
    hardware-model-label => hardware-model-type
```

```
)

hardware-model-type = JC<base64-url-text .size (4..44),
                         bytes .size (1..32)>

$$Claims-Set-Claims //= ( sw-name-label => tstr )

$$Claims-Set-Claims //= (sw-version-label => sw-version-type)

sw-version-type = [
    version:  tstr
    ? scheme:  $version-scheme
]

$$Claims-Set-Claims //= (secure-boot-label => bool)

$$Claims-Set-Claims //= ( debug-status-label => debug-status-type )

debug-status-type = ds-enabled /
                    disabled /
                    disabled-since-boot /
                    disabled-permanently /
                    disabled-fully-and-permanently

ds-enabled                    = JC< "enabled", 0 >
disabled                      = JC< "disabled", 1 >
disabled-since-boot           = JC< "disabled-since-boot", 2 >
disabled-permanently          = JC< "disabled-permanently", 3 >
disabled-fully-and-permanently = JC< "disabled-fully-and-permanently",
                                     4 >

$$Claims-Set-Claims //= (location-label => location-type)

location-type = {
    latitude => number,
    longitude => number,
    ? altitude => number,
    ? accuracy => number,
    ? altitude-accuracy => number,
    ? heading => number,
    ? speed => number,
    ? timestamp => ~time-int,
    ? age => uint
}

latitude          = JC< "latitude",          1 >
longitude         = JC< "longitude",         2 >
altitude          = JC< "altitude",          3 >
accuracy          = JC< "accuracy",          4 >
altitude-accuracy = JC< "altitude-accuracy", 5 >
heading           = JC< "heading",           6 >
speed             = JC< "speed",             7 >
timestamp         = JC< "timestamp",         8 >
age               = JC< "age",               9 >

$$Claims-Set-Claims //= (uptime-label => uint)

$$Claims-Set-Claims //=  (boot-seed-label => binary-data)
```

```
$$Claims-Set-Claims //= (boot-count-label => uint)

$$Claims-Set-Claims //= ( intended-use-label => intended-use-type )

intended-use-type = generic /
                    registration /
                    provisioning /
                    csr /
                    pop

generic      = JC< "generic",      1 >
registration = JC< "registration", 2 >
provisioning = JC< "provisioning", 3 >
csr          = JC< "csr",          4 >
pop          = JC< "pop",          5 >

$$Claims-Set-Claims //= (
    dloas-label => [ + dloa-type ]
)

dloa-type = [
    dloa_registrar: general-uri
    dloa_platform_label: text
    ? dloa_application_label: text
]

$$Claims-Set-Claims //= (profile-label => general-uri / general-oid)

$$Claims-Set-Claims //= (
    manifests-label => manifests-type
)

manifests-type = [+ manifest-format]

manifest-format = [
    content-type:   coap-content-format,
    content-format: JC< $manifest-body-json,
                        $manifest-body-cbor >
]

$manifest-body-cbor /= bytes .cbor untagged-coswid
$manifest-body-json /= base64-url-text

$manifest-body-cbor /= bytes .cbor SUIT_Envelope
$manifest-body-json /= base64-url-text

$manifest-body-cbor /= spdx-json
$manifest-body-json /= spdx-json

spdx-json = text

$manifest-body-cbor /= cyclone-dx-json
$manifest-body-cbor /= cyclone-dx-xml
$manifest-body-json /= cyclone-dx-json
$manifest-body-json /= cyclone-dx-xml
cyclone-dx-json = text
cyclone-dx-xml  = text
```

```
suit-directive-process-dependency = 19

$$Claims-Set-Claims //= (
    measurements-label => measurements-type
)

measurements-type = [+ measurements-format]

measurements-format = [
    content-type:   coap-content-format,
    content-format: JC< $$measurements-body-json,
                        $$measurements-body-cbor >
]

$$measurements-body-cbor /= bytes .cbor untagged-coswid
$$measurements-body-json /= base64-url-text


$$Claims-Set-Claims //= (
    measurement-results-label =>
        [ + measurement-results-group ] )

measurement-results-group = [
    measurement-system: tstr,
    measruement-results: [ + individual-result ]
]

individual-result = [
    results-id: tstr / binary-data,
    result:     result-type,
]

result-type = comparison-successful /
              comparison-fail /
              comparison-not-run /
              measurement-absent

comparison-successful    = JC< "success",       1 >
comparison-fail          = JC< "fail",          2 >
comparison-not-run       = JC< "not-run",       3 >
measurement-absent       = JC< "absent",        4 >



$$Claims-Set-Claims //= (submods-label => { + text => Submodule })

Submodule = Claims-Set / Nested-Token / Detached-Submodule-Digest



Detached-Submodule-Digest = [
   hash-algorithm : text / int,
   digest         : binary-data
]


BUNDLE-Messages = BUNDLE-Tagged-Message / BUNDLE-Untagged-Message
```

```
BUNDLE-Tagged-Message   = #6.TBD(BUNDLE-Untagged-Message)
BUNDLE-Untagged-Message = Detached-EAT-Bundle

Detached-EAT-Bundle = [
    main-token : Nested-Token,
    detached-claims-sets: {
        + tstr => JC<json-wrapped-claims-set,
                     cbor-wrapped-claims-set>
    }
]

json-wrapped-claims-set = base64-url-text

cbor-wrapped-claims-set = bstr .cbor Claims-Set



nonce-label            = JC< "eat_nonce",  10 >
ueid-label             = JC< "ueid",       256 >
sueids-label           = JC< "sueids",     257 >
oemid-label            = JC< "oemid",      258 >
hardware-model-label   = JC< "hwmodel",    259 >
hardware-version-label = JC< "hwversion",  260 >
secure-boot-label      = JC< "secboot",    262 >
debug-status-label     = JC< "dbgstat",    263 >
location-label         = JC< "location",   264 >
profile-label          = JC< "eat_profile",265 >
submods-label          = JC< "submods",    266 >

uptime-label           = JC< "uptime",     TBD >
boot-seed-label        = JC< "bootseed",   TBD >
intended-use-label     = JC< "intuse",     TBD >
dloas-label            = JC< "dloas",      TBD >
sw-name-label          = JC< "swname",     TBD >
sw-version-label       = JC< "swversion",  TBD >
manifests-label        = JC< "manifests",  TBD >
measurements-label     = JC< "measurements", TBD >
measurement-results-label = JC< "measres" , TBD >
boot-count-label       = JC< "bootcount",  TBD >
```

7.3.2. CBOR-Specific CDDL

```
EAT-CBOR-Token = $$EAT-CBOR-Tagged-Token / $$EAT-CBOR-Untagged-Token

$$EAT-CBOR-Tagged-Token /= CWT-Tagged-Message
$$EAT-CBOR-Tagged-Token /= BUNDLE-Tagged-Message

$$EAT-CBOR-Untagged-Token /= CWT-Untagged-Message
$$EAT-CBOR-Untagged-Token /= BUNDLE-Untagged-Message


Nested-Token = CBOR-Nested-Token

CBOR-Nested-Token =
    JSON-Token-Inside-CBOR-Token /
    CBOR-Token-Inside-CBOR-Token

CBOR-Token-Inside-CBOR-Token = bstr .cbor $$EAT-CBOR-Tagged-Token

JSON-Token-Inside-CBOR-Token = tstr
```

7.3.3. JSON-Specific CDDL

```
EAT-JSON-Token = $$EAT-JSON-Token-Formats

$$EAT-JSON-Token-Formats /= JWT-Message
$$EAT-JSON-Token-Formats /= BUNDLE-Untagged-Message


Nested-Token = JSON-Nested-Token

JSON-Nested-Token = [
   type : "JWT" / "CBOR" / "BUNDLE",
   nested-token : JWT-Message /
                  CBOR-Token-Inside-JSON-Token /
                  Detached-EAT-Bundle
]

CBOR-Token-Inside-JSON-Token = base64-url-text
```

8. Privacy Considerations

Certain EAT claims can be used to track the owner of an entity and therefore, implementations should consider providing privacy-preserving options dependent on the intended usage of the EAT. Examples would include suppression of location claims for EAT's provided to unauthenticated consumers.

8.1. UEID and SUEID Privacy Considerations

A UEID is usually not privacy-preserving. Any set of Relying Parties that receives tokens that happen to be from a particular entity will be able to know the tokens are all from the same entity and be able to track it.

Thus, in many usage situations UEID violates governmental privacy regulation. In other usage situations a UEID will not be allowed for certain products like browsers that give privacy for the end user. It will often be the case that tokens will not have a UEID for these reasons.

An SUEID is also usually not privacy-preserving. In some cases it may have fewer privacy issues than a UEID depending on when and how and when it is generated.

There are several strategies that can be used to still be able to put UEIDs and SUEIDs in tokens:

- The entity obtains explicit permission from the user of the entity to use the UEID/SUEID. This may be through a prompt. It may also be through a license agreement. For example, agreements for some online banking and brokerage services might already cover use of a UEID/SUEID.
- The UEID/SUEID is used only in a particular context or particular use case. It is used only by one relying party.
- The entity authenticates the relying party and generates a derived UEID/SUEID just for that particular relying party. For example, the Relying Party could prove their identity cryptographically to the entity, then the entity generates a UEID just for that relying party by hashing a proofed relying party ID with the main entity UEID/SUEID.

Note that some of these privacy preservation strategies result in multiple UEIDs and SUEIDs per entity. Each UEID/SUEID is used in a different context, use case or system on the entity. However, from the view of the relying party, there is just one UEID and it is still globally universal across manufacturers.

8.2. Location Privacy Considerations

Geographic location is most always considered personally identifiable information. Implementers should consider laws and regulations governing the transmission of location data from end user devices to servers and services. Implementers should consider using location management facilities offered by the operating system on the entity generating the attestation. For example, many mobile phones prompt the user for permission when before sending location data.

8.3. Boot Seed Privacy Considerations

The "bootseed" claim is effectively a stable entity identifier within a given boot epoch. Therefore, it is not suitable for use in attestation schemes that are privacy-preserving.

8.4. Replay Protection and Privacy

EAT defines the nonce claim for token replay protection (also sometimes known as token "freshness"). The nonce claim is based on a value that is usually derived remotely (outside of the entity). This claim can be used to extract and convey personally-identifying information either inadvertently or by intention. For instance, an implementor may choose a nonce that is equivalent to a username associated with the device (e.g., account login). If the token is inspected by a 3rd-party then this information could be used to identify the source of the token or an account associated with the token. In order to avoid the conveyance of privacy-related information in the nonce claim, it should be derived using a salt that originates from a true and reliable random number generator or any other source of randomness that would still meet the target system requirements for replay protection.

9. Security Considerations

The security considerations provided in Section 8 of [RFC8392] and Section 11 of [RFC7519] apply to EAT in its CWT and JWT form, respectively. Moreover, Chapter 12 of [RATS.Architecture] is also applicable to implementations of EAT. In addition, implementors should consider the following.

9.1. Claim Trustworthiness

This specification defines semantics for each claim. It does not require any particular level of security in the implementation of the claims or even the attester itself. Such specification is far beyond the scope of this document which is about a message format not the security level of an implementation.

The receiver of an EAT comes to know the trustworthiness of the claims in it by understanding the implementation made by the attester vendor and/or understanding the checks and processing performed by the verifier.

For example, this document says that a UEID is permanent and that it must not change, but it doesn't say what degree of attack to change it must be defended.

The degree of security will vary from use case to use case. In some cases the receiver may only need to know something of the implementation such as that it was implemented in a TEE. In other cases the receiver may require the attester be certified by a particular certification program. Or perhaps the receiver is content with very little security.

9.2. Key Provisioning

Private key material can be used to sign and/or encrypt the EAT, or can be used to derive the keys used for signing and/or encryption. In some instances, the manufacturer of the entity may create the key material separately and provision the key material in the entity itself. The manufacturer of any entity that is capable of producing an EAT should take care to ensure that any private key material be suitably protected prior to provisioning the key material in the entity itself. This can require creation of key material in an enclave (see [RFC4949] for definition of "enclave"), secure

transmission of the key material from the enclave to the entity using an appropriate protocol, and persistence of the private key material in some form of secure storage to which (preferably) only the entity has access.

9.2.1. Transmission of Key Material

Regarding transmission of key material from the enclave to the entity, the key material may pass through one or more intermediaries. Therefore some form of protection ("key wrapping") may be necessary. The transmission itself may be performed electronically, but can also be done by human courier. In the latter case, there should be minimal to no exposure of the key material to the human (e.g. encrypted portable memory). Moreover, the human should transport the key material directly from the secure enclave where it was created to a destination secure enclave where it can be provisioned.

9.3. Freshness

All EAT use must provide a freshness mechanism to prevent replay and related attacks. The extensive discussions on freshness in [RATS.Architecture] including security considerations apply here. The EAT nonce claim, in Section 4.1, is one option to provide freshness.

9.4. Multiple EAT Consumers

In many cases, more than one EAT consumer may be required to fully verify the entity attestation. Examples include individual consumers for nested EATs, or consumers for individual claims with an EAT. When multiple consumers are required for verification of an EAT, it is important to minimize information exposure to each consumer. In addition, the communication between multiple consumers should be secure.

For instance, consider the example of an encrypted and signed EAT with multiple claims. A consumer may receive the EAT (denoted as the "receiving consumer"), decrypt its payload, verify its signature, but then pass specific subsets of claims to other consumers for evaluation ("downstream consumers"). Since any COSE encryption will be removed by the receiving consumer, the communication of claim subsets to any downstream consumer should leverage a secure protocol (e.g.one that uses transport-layer security, i.e. TLS),

However, assume the EAT of the previous example is hierarchical and each claim subset for a downstream consumer is created in the form of a nested EAT. Then transport security between the receiving and downstream consumers is not strictly required. Nevertheless, downstream consumers of a nested EAT should provide a nonce unique to the EAT they are consuming.

9.5. Detached EAT Bundle Digest Security Considerations

A detached EAT bundle is composed of a nested full token appended to an unsigned claims set as per Section 5 . Although the attached claims set is vulnerable to modification in transit, any modification can be detected by the receiver through the associated digest, which is a claim fully contained within an EAT. Moreover, the digest itself can only be derived using an appropriate

COSE hash algorithm, implying that an attacker cannot induce false detection of a modifie detached claims because the algorithms in the COSE registry are assumed to be of sufficient cryptographic strength.

10. IANA Considerations

10.1. Reuse of CBOR and JSON Web Token (CWT and JWT) Claims Registries

Claims defined for EAT are compatible with those of CWT and JWT so the CWT and JWT Claims Registries, [IANA.CWT.Claims] and [IANA.JWT.Claims], are re used. No new IANA registry is created.

All EAT claims defined in this document are placed in both registries. All new EAT claims defined subsequently should be placed in both registries.

Appendix E describes some considerations when defining new claims.

10.2. Claims Registered by This Document

This specification adds the following values to the "JSON Web Token Claims" registry established by [RFC7519] and the "CBOR Web Token Claims Registry" established by [RFC8392]. Each entry below is an addition to both registries.

The "Claim Description", "Change Controller" and "Specification Documents" are common and equivalent for the JWT and CWT registries. The "Claim Key" and "Claim Value Types(s)" are for the CWT registry only. The "Claim Name" is as defined for the CWT registry, not the JWT registry. The "JWT Claim Name" is equivalent to the "Claim Name" in the JWT registry.

10.2.1. Claims for Early Assignment

RFC Editor: in the final publication this section should be combined with the following section as it will no longer be necessary to distinguish claims with early assignment. Also, the following paragraph should be removed.

The claims in this section have been (requested for / given) early assignment according to [RFC7120]. They have been assigned values and registered before final publication of this document. While their semantics is not expected to change in final publication, it is possible that they will. The JWT Claim Names and CWT Claim Keys are not expected to change.

In draft -06 an early allocation was described. The processing of that early allocation was never correctly completed. This early allocation assigns different numbers for the CBOR claim labels. This early allocation will presumably complete correctly

- Claim Name: Nonce
- Claim Description: Nonce
- JWT Claim Name: "eat_nonce"
- Claim Key: TBD (requested value 10)

- Claim Value Type(s): byte string
- Change Controller: IESG
- Specification Document(s): **this document**

- Claim Name: UEID
- Claim Description: The Universal Entity ID
- JWT Claim Name: "ueid"
- CWT Claim Key: TBD (requested value 256)
- Claim Value Type(s): byte string
- Change Controller: IESG
- Specification Document(s): **this document**

- Claim Name: SUEIDs
- Claim Description: Semi-permanent UEIDs
- JWT Claim Name: "sueids"
- CWT Claim Key: TBD (requested value 257)
- Claim Value Type(s): map
- Change Controller: IESG
- Specification Document(s): **this document**

- Claim Name: Hardware OEMID
- Claim Description: Hardware OEM ID
- JWT Claim Name: "oemid"
- Claim Key: TBD (requeste value 258)
- Claim Value Type(s): byte string or integer
- Change Controller: IESG
- Specification Document(s): **this document**

- Claim Name: Hardware Model
- Claim Description: Model identifier for hardware
- JWT Claim Name: "hwmodel"
- Claim Key: TBD (requested value 259)
- Claim Value Type(s): byte string
- Change Controller: IESG
- Specification Document(s): **this document**

- Claim Name: Hardware Version
- Claim Description: Hardware Version Identifier
- JWT Claim Name: "hwversion"
- Claim Key: TBD (requested value 260)
- Claim Value Type(s): array

- Change Controller: IESG
- Specification Document(s): **this document**

- Claim Name: Secure Boot
- Claim Description: Indicate whether the boot was secure
- JWT Claim Name: "secboot"
- Claim Key: 262
- Claim Value Type(s): Boolean
- Change Controller: IESG
- Specification Document(s): **this document**

- Claim Name: Debug Status
- Claim Description: Indicate status of debug facilities
- JWT Claim Name: "dbgstat"
- Claim Key: 263
- Claim Value Type(s): integer or string
- Change Controller: IESG
- Specification Document(s): **this document**

- Claim Name: Location
- Claim Description: The geographic location
- JWT Claim Name: "location"
- Claim Key: TBD (requested value 264)
- Claim Value Type(s): map
- Change Controller: IESG
- Specification Document(s): **this document**

- Claim Name: EAT Profile
- Claim Description: Indicates the EAT profile followed
- JWT Claim Name: "eat_profile"
- Claim Key: TBD (requested value 265)
- Claim Value Type(s): URI or OID
- Change Controller: IESG
- Specification Document(s): **this document**

- Claim Name: Submodules Section
- Claim Description: The section containing submodules
- JWT Claim Name: "submods"
- Claim Key: TBD (requested value 266)
- Claim Value Type(s): map
- Change Controller: IESG

- Specification Document(s): **this document**

10.2.2. To be Assigned Claims

(Early assignment is NOT requested for these claims. Implementers should be aware they may change)

- Claim Name: Uptime
- Claim Description: Uptime
- JWT Claim Name: "uptime"
- Claim Key: TBD
- Claim Value Type(s): unsigned integer
- Change Controller: IESG
- Specification Document(s): **this document**

- Claim Name: Boot Seed
- Claim Description: Identifies a boot cycle
- JWT Claim Name: "bootseed"
- Claim Key: TBD
- Claim Value Type(s): bytes
- Change Controller: IESG
- Specification Document(s): **this document**

- Claim Name: Intended Use
- Claim Description: Indicates intended use of the EAT
- JWT Claim Name: "intuse"
- Claim Key: TBD
- Claim Value Type(s): integer or string
- Change Controller: IESG
- Specification Document(s): **this document**

- Claim Name: DLOAs
- Claim Description: Certifications received as Digital Letters of Approval
- JWT Claim Name: "dloas"
- Claim Key: TBD
- Claim Value Type(s): array
- Change Controller: IESG
- Specification Document(s): **this document**

- Claim Name: Software Name
- Claim Description: The name of the software running in the entity
- JWT Claim Name: "swname"
- Claim Key: TBD

- Claim Value Type(s): map
- Change Controller: IESG
- Specification Document(s): **this document**

- Claim Name: Software Version
- Claim Description: The version of software running in the entity
- JWT Claim Name: "swversion"
- Claim Key: TBD
- Claim Value Type(s): map
- Change Controller: IESG
- Specification Document(s): **this document**

- Claim Name: Software Manifests
- Claim Description: Manifests describing the software installed on the entity
- JWT Claim Name: "manifests"
- Claim Key: TBD
- Claim Value Type(s): array
- Change Controller: IESG
- Specification Document(s): **this document**

- Claim Name: Measurements
- Claim Description: Measurements of the software, memory configuration and such on the entity
- JWT Claim Name: "measurements"
- Claim Key: TBD
- Claim Value Type(s): array
- Change Controller: IESG
- Specification Document(s): **this document**

- Claim Name: Software Measurement Results
- Claim Description: The results of comparing software measurements to reference values
- JWT Claim Name: "measres"
- Claim Key: TBD
- Claim Value Type(s): array
- Change Controller: IESG
- Specification Document(s): **this document**

- Claim Name: Boot Count
- Claim Description: The number times the entity or submodule has been booted
- JWT Claim Name: "bootcount"
- Claim Key: TBD

- Claim Value Type(s): uint
- Change Controller: IESG
- Specification Document(s): **this document**

10.2.3. UEID URN Registered by this Document

IANA is requested to register the following new subtypes in the "DEV URN Subtypes" registry under "Device Identification". See [RFC9039].

Subtype	Description	Reference
ueid	Universal Entity Identifier	This document
sueid	Semi-permanent Universal Entity Identifier	This document

Table 3

10.2.4. Tag for Detached EAT Bundle

In the registry [IANA.cbor-tags], IANA is requested to allocate the following tag from the FCFS space, with the present document as the specification reference.

Tag	Data Items	Semantics
TBD602	array	Detached EAT Bundle Section 5

Table 4

10.2.5. Media Types Registered by this Document

It is requested that the CoAP Content-Format for SPDX and CycloneDX be been registered in the "CoAP Content-Formats" subregistry within the "Constrained RESTful Environments (CoRE) Parameters" registry [IANA.core-parameters]:

- Media Type: application/spdx+json
- Encoding: binary
- ID: TBD
- Reference: [SPDX]

- Media Type: vendor/vnd.cyclonedx+xml
- Encoding: binary
- ID: TBD
- Reference: [CycloneDX]

- Media Type: vendor/vnd.cyclonedx+json
- Encoding: binary
- ID: TBD
- Reference: [CycloneDX]

11. References

11.1. Normative References

[CoSWID] Birkholz, H., Fitzgerald-McKay, J., Schmidt, C., and D. Waltermire, "Concise Software Identification Tags", Work in Progress, Internet-Draft, draft-ietf-sacm-coswid-22, 20 July 2022, <https://www.ietf.org/archive/id/draft-ietf-sacm-coswid-22.txt>.

[CycloneDX] "CycloneDX", <https://cyclonedx.org/specification/overview/>.

[DLOA] "Digital Letter of Approval", November 2015, <https://globalplatform.org/wp-content/uploads/2015/12/GPC_DigitalLetterOfApproval_v1.0.pdf>.

[IANA.cbor-tags] "IANA CBOR Tags Registry", n.d., <https://www.iana.org/assignments/cbor-tags/cbor-tags.xhtml>.

[IANA.core-parameters] "IANA Constrained RESTful Environments (CoRE) Parameters", n.d., <<https://www.iana.org/assignments/core-parameters>>.

[IANA.COSE.Algorithms] "COSE Algorithms", <https://www.iana.org/assignments/cose/>.

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

[IANA.JWT.Claims] IANA, "JSON Web Token (JWT) Claims", <https://www.iana.org/assignments/jwt>.

[PEN] "Private Enterprise Number (PEN) Request", n.d., <https://pen.iana.org/pen/PenApplication.page>.

[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>.

[RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <https://www.rfc-editor.org/info/rfc7252>.

[RFC7515] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Signature (JWS)", RFC 7515, DOI 10.17487/RFC7515, May 2015, <https://www.rfc-editor.org/info/rfc7515>.

[RFC7519] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Token (JWT)", RFC 7519, DOI 10.17487/RFC7519, May 2015, <https://www.rfc-editor.org/info/rfc7519>.

[RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <https://www.rfc-editor.org/info/rfc8126>.

[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>.

[RFC8259] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", STD 90, RFC 8259, DOI 10.17487/RFC8259, December 2017, <https://www.rfc-editor.org/info/rfc8259>.

[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>.

[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/info/rfc8949>.

[RFC9052] 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>.

[RFC9090] Bormann, C., "Concise Binary Object Representation (CBOR) Tags for Object Identifiers", RFC 9090, DOI 10.17487/RFC9090, July 2021, <https://www.rfc-editor.org/info/rfc9090>.

[SPDX] "Software Package Data Exchange (SPDX)", 2020, <https://spdx.dev/wp-content/uploads/sites/41/2020/08/SPDX-specification-2-2.pdf>.

[SUIT.Manifest] Moran, B., Tschofenig, H., Birkholz, H., and K. Zandberg, "A Concise Binary Object Representation (CBOR)-based Serialization Format for the Software Updates for Internet of Things (SUIT) Manifest", Work in Progress, Internet-Draft, draft-ietf-suit-manifest-19, 9 August 2022, <https://www.ietf.org/archive/id/draft-ietf-suit-manifest-19.txt>.

[ThreeGPP.IMEI] 3GPP, "3rd Generation Partnership Project; Technical Specification Group Core Network and Terminals; Numbering, addressing and identification", 2019, <https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=729>.

[WGS84] National Geospatial-Intelligence Agency (NGA), "WORLD GEODETIC SYSTEM 1984, NGA.STND.0036_1.0.0_WGS84", 8 July 2014, <https://earth-info.nga.mil/php/download.php?file=coord-wgs84>.

11.2. Informative References

[BirthdayAttack] "Birthday attack", <https://en.wikipedia.org/wiki/Birthday_attack.>.

[CBOR.Cert.Draft] Mattsson, J. P., Selander, G., Raza, S., Höglund, J., and M. Furuhed, "CBOR Encoded X.509 Certificates (C509 Certificates)", Work in Progress, Internet-Draft, draft-ietf-cose-cbor-encoded-cert-04, 10 July 2022, <https://www.ietf.org/archive/id/draft-ietf-cose-cbor-encoded-cert-04.txt>.

[COSE.X509.Draft] Schaad, J., "CBOR Object Signing and Encryption (COSE): Header parameters for carrying and referencing X.509 certificates", Work in Progress, Internet-Draft, draft-ietf-cose-x509-08, 14 December 2020, <https://www.ietf.org/archive/id/draft-ietf-cose-x509-08.txt>.

[IEEE.802-2001] "IEEE Standard For Local And Metropolitan Area Networks Overview And Architecture", 2007, <https://webstore.ansi.org/standards/ieee/ieee8022001r2007>.

[IEEE.802.1AR] "IEEE Standard, "IEEE 802.1AR Secure Device Identifier"", December 2009, <http://standards.ieee.org/findstds/standard/802.1AR-2009.html>.

[IEEE.RA] "IEEE Registration Authority", <https://standards.ieee.org/products-services/regauth/index.html>.

[OUI.Guide] "Guidelines for Use of Extended Unique Identifier (EUI), Organizationally Unique Identifier (OUI), and Company ID (CID)", August 2017, <https://standards.ieee.org/content/dam/ieee-standards/standards/web/documents/tutorials/eui.pdf>.

[OUI.Lookup] "IEEE Registration Authority Assignments", <https://regauth.standards.ieee.org/standards-ra-web/pub/view.html#registries>.

[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-22, 28 September 2022, <https://www.ietf.org/archive/id/draft-ietf-rats-architecture-22.txt>.

[RFC4122] Leach, P., Mealling, M., and R. Salz, "A Universally Unique IDentifier (UUID) URN Namespace", RFC 4122, DOI 10.17487/RFC4122, July 2005, <https://www.rfc-editor.org/info/rfc4122>.

[RFC4949] Shirey, R., "Internet Security Glossary, Version 2", FYI 36, RFC 4949, DOI 10.17487/RFC4949, August 2007, <https://www.rfc-editor.org/info/rfc4949>.

[RFC7120] Cotton, M., "Early IANA Allocation of Standards Track Code Points", BCP 100, RFC 7120, DOI 10.17487/RFC7120, January 2014, <https://www.rfc-editor.org/info/rfc7120>.

[RFC9039] Arkko, J., Jennings, C., and Z. Shelby, "Uniform Resource Names for Device Identifiers", RFC 9039, DOI 10.17487/RFC9039, June 2021, <https://www.rfc-editor.org/info/rfc9039>.

[UCCS] Birkholz, H., O'Donoghue, J., Cam-Winget, N., and C. Bormann, "A CBOR Tag for Unprotected CWT Claims Sets", Work in Progress, Internet-Draft, draft-ietf-rats-uccs-03, 11 July 2022, <https://www.ietf.org/archive/id/draft-ietf-rats-uccs-03.txt>.

[W3C.GeoLoc] Worldwide Web Consortium, "Geolocation API Specification 2nd Edition", January 2018, <https://www.w3.org/TR/geolocation-API/#coordinates_interface>.

Appendix A. Examples

Most examples are shown as just a Claims-Set that would be a payload for a CWT, JWT, detached EAT bundle or future token types. It is shown this way because the payload is all the claims, the most interesting part and showing full tokens makes it harder to show the claims.

Some examples of full tokens are also given.

WARNING: These examples use tag and label numbers not yet assigned by IANA.

A.1. Payload Examples

A.1.1. Simple TEE Attestation

This is a simple attestation of a TEE that includes a manifest that is a payload CoSWID to describe the TEE's software.

```
/ This is an EAT payload that describes a simple TEE. /

{
    / eat_nonce /       10: h'948f8860d13a463e',
    / secboot /        262: true,
    / dbgstat /        263: 2, / disabled-since-boot /
    / manifests /      273: [
                            [
                                121, / CoAP Content ID. A     /
                                     / made up one until one  /
                                     / is assigned for CoSWID /

                                / This is byte-string wrapped      /
                                / payload CoSWID. It gives the TEE /
                                / software name, the version and   /
                                / the  name of the file it is in.  /
                                / {0: "3a24",                      /
                                /  12: 1,                          /
                                /   1: "Acme TEE OS",              /
                                /  13: "3.1.4",                    /
                                /   2: [{31: "Acme TEE OS", 33: 1}, /
                                /       {31: "Acme TEE OS", 33: 2}], /
                                /   6: {                           /
                                /       17: {                      /
                                /           24: "acme_tee_3.exe"   /
                                /       }                          /
                                /    }                             /
                                /  }                               /
                                h' a60064336132340c01016b
                                   41636d6520544545204f530d65332e31
                                   2e340282a2181f6b41636d6520544545
                                   204f53182101a2181f6b41636d652054
                                   4545204f5318210206a111a118186e61
                                   636d655f7465655f332e657865'
                            ]
                        ]
}
```

```
/ A payload CoSWID created by the SW vendor. All this really does /
/ is name the TEE SW, its version and lists the one file that     /
/ makes up the TEE. /

1398229316({
    / Unique CoSWID ID /    0: "3a24",
    / tag-version /        12: 1,
    / software-name /       1: "Acme TEE OS",
    / software-version /   13: "3.1.4",
    / entity /              2: [
                                   {
        / entity-name /                31: "Acme TEE OS",
        / role        /                33: 1 / tag-creator /
                                   },
                                   {
        / entity-name /                31: "Acme TEE OS",
        / role        /                33: 2 / software-creator /
                                   }
                               ],
    / payload /                6: {
        / ...file /                17: {
            / ...fs-name /             24: "acme_tee_3.exe"
                                   }
                               }
})
```

A.1.2. Submodules for Board and Device

```
/ This example shows use of submodules to give information  /
/ about the chip, board and overall device.                 /
/                                                           /
/ The main attestation is associated with the chip with the /
/ CPU and running the main OS. It is what has the keys and  /
/ produces the token.                                       /
/                                                           /
/ The board is made by a different vendor than the chip.    /
/ Perhaps it is some generic IoT board.                     /
/                                                           /
/ The device is some specific appliance that is made by a   /
/ different vendor than either the chip or the board.       /
/                                                           /
/ Here the board and device submodules aren't the typical   /
/ target environments as described by the RATS architecture /
/ document, but they are a valid use of submodules.         /

{
    / eat_nonce /       10: h'948f8860d13a463e8e',
    / ueid /           256: h'0198f50a4ff6c05861c8860d13a638ea',
    / oemid /          258: h'894823', / IEEE OUI format OEM ID /
    / hwmodel /        259: h'549dcecc8b987c737b44e40f7c635ce8'
                              / Hash of chip model name /,
    / hwversion /      260: ["1.3.4", 1], / Multipartnumeric version /
    / swname /         271: "Acme OS",
    / swversion /      272: ["3.5.5", 1],
    / secboot /        262: true,
    / dbgstat /        263: 3, / permanent-disable  /
    / timestamp (iat) /  6: 1526542894,
    / submods / 266: {
        / A submodule to hold some claims about the circuit board /
        "board" :  {
            / oemid /       258: h'9bef8787eba13e2c8f6e7cb4b1f4619a',
            / hwmodel /     259: h'ee80f5a66c1fb9742999a8fdab930893'
                                    / Hash of board module name /,
            / hwversion /   260: ["2.0a", 2] / multipartnumeric+suffix /
        },

        / A submodule to hold claims about the overall device /
        "device" :  {
            / oemid /       258: 61234, / PEN Format OEM ID /
            / hwversion /   260: ["4.0", 1] / Multipartnumeric version /
        }
    }
}
```

A.1.3. EAT Produced by Attestation Hardware Block

```
/ This is an example of a token produced by a HW block            /
/ purpose-built for attestation.  Only the nonce claim changes    /
/ from one attestation to the next as the rest  either come       /
/ directly from the hardware or from one-time-programmable memory /
/ (e.g. a fuse). 47 bytes encoded in CBOR (8 byte nonce, 16 byte  /
/ UEID). /

{
    / eat_nonce /       10: h'948f8860d13a463e',
    / ueid /           256: h'0198f50a4ff6c05861c8860d13a638ea',
    / oemid /          258: 64242, / Private Enterprise Number /
    / secboot /        262: true,
    / dbgstat /        263: 3, / disabled-permanently /
    / hwversion /      260: [ "3.1", 1 ] / Type is multipartnumeric /
}
```

A.1.4. Key / Key Store Attestation

```
/ This is an attestation of a public key and the key store     /
/ implementation that protects and manages it. The key store   /
/ implementation is in a security-oriented execution           /
/ environment separate from the high-level OS, for example a   /
/ TEE. The key store is the Attester.                          /
/                                                              /
/ There is some attestation of the high-level OS, just version /
/ and boot & debug status. It is a Claims-Set submodule because/
/ it has lower security level than the key store. The key      /
/ store's implementation has access to info about the HLOS, so /
/ it is able to include it.                                    /
/                                                              /
/ A key and an indication of the user authentication given to  /
/ allow access to the key is given. The labels for these are   /
/ in the private space since this is just a hypothetical       /
/ example, not part of a standard protocol.                    /
/                                                              /
/ This is similar to Android Key Attestation.                  /


{
    / eat_nonce /       10: h'948f8860d13a463e',
    / secboot /        262: true,
    / dbgstat /        263: 2, / disabled-since-boot /
    / manifests /      273: [
                               [ 121, / CoAP Content ID. A      /
                                      / made up one until one   /
                                      / is assigned for CoSWID  /
                                 h'a600683762623334383766
                                   0c000169436172626f6e6974650d6331
                                   2e320e0102a2181f75496e6475737472
                                   69616c204175746f6d6174696f6e1821
                                   02'
                                ]
                                / Above is an encoded CoSWID     /
                                / with the following data        /
                                /   SW Name: "Carbonite"         /
                                /   SW Vers: "1.2"               /
                                /   SW Creator:                  /
                                /      "Industrial Automation"   /
                            ],
    / exp /              4: 1634324274, / 2021-10-15T18:57:54Z /
    / iat /              6: 1634317080, / 2021-10-15T16:58:00Z /
                  -80000 : "fingerprint",
                  -80001 : { / The key -- A COSE_Key  /
                / kty /       1: 2, / EC2, eliptic curve with x & y /
                / kid /       2: h'36675c206f96236c3f51f54637b94ced',
                / curve /    -1: 2, / curve is P-256 /
                / x-coord /  -2: h'65eda5a12577c2bae829437fe338701a
                                   10aaa375e1bb5b5de108de439c08551d',
                / y-coord /  -3: h'1e52ed75701163f7f9e40ddf9f341b3d
                                   c9ba860af7e0ca7ca7e9eecd0084d19c'
             },

    / submods /        266 : {
                           "HLOS" : { / submod for high-level OS /
         / eat_nonce /         10: h'948f8860d13a463e',
```

```
            / secboot /        262: true,
            / manifests /      273: [
                                 [ 121, / CoAP Content ID. A      /
                                        / made up one until one   /
                                        / is assigned for CoSWID  /
                                     h'a600687337
                                       6537346b78380c000168
                                       44726f6964204f530d65
                                       52322e44320e0302a218
                                       1F75496E647573747269
                                       616c204175746f6d6174
                                       696f6e182102'
                                   ]
                                   / Above is an encoded CoSWID /
                                   / with the following data:   /
                                   /   SW Name: "Droid OS"      /
                                   /   SW Vers: "R2.D2"         /
                                   /   SW Creator:              /
                                   /     "Industrial Automation"/
                               ]
                           }
                       }
}
```

A.1.5. Software Measurements of an IoT Device

This is a simple token that might be for and IoT device. It includes CoSWID format measurments of the SW. The CoSWID is in byte-string wrapped in the token and also shown in diagnostic form.

```
/ This EAT payload is for an IoT device with a TEE. The attestation /
/ is produced by the TEE. There is a submodule for the IoT OS (the  /
/ main OS of the IoT device that is not as secure as the TEE). The  /
/ submodule contains claims for the IoT OS. The TEE also measures   /
/ the IoT OS and puts the measurements in the submodule.            /

{
    / eat_nonce /       10: h'948f8860d13a463e',
    / secboot /        262: true,
    / dbgstat /        263: 2, / disabled-since-boot /
    / oemid /          258: h'8945ad', / IEEE CID based /
    / ueid /           256: h'0198f50a4ff6c05861c8860d13a638ea',
    / submods /        266: {
                            "OS" : {
        / secboot /            262: true,
        / dbgstat /            263: 2, / disabled-since-boot /
        / measurements         274: [
                                    [
                                        121, / CoAP Content ID. A     /
                                             / made up one until one  /
                                             / is assigned for CoSWID /

                                     / This is a byte-string wrapped /
                                     / evidence CoSWID. It has       /
                                     / hashes of the main files of   /
                                     / the IoT OS.  /
                                     h'a600663463613234350c
                                       17016d41636d6520522d496f542d4f
                                       530d65332e312e3402a2181f724163
                                       6d652042617365204174746573746 5
                                       7218210103a11183a318187161636d
                                       655f725f696f745f6f732e65786514
                                       1a0044b349078201582005f6b327c1
                                       73b4192bd2c3ec248a292215eab456
                                       611bf7a783e25c1782479905a31818
                                       6d7265736f75726365732e72736314
                                       1a000c38b10782015820c142b9aba4
                                       280c4bb8c75f716a43c99526694caa
                                       be529571f5569bb7dc542f98a31818
                                       6a636f6d6d6f6e2e6c6962141a0023
                                       3d3b0782015820a6a9dcdfb3884da5
                                       f884e4e1e8e8629958c2dbc7027414
                                       43a913e34de9333be6'
                                    ]
                                ]
                            }
                        }
}
```

```
/ An evidence CoSWID created for the "Acme R-IoT-OS" created by /
/ the "Acme Base Attester" (both fictious names).  It provides  /
/ measurements of the SW (other than the attester SW) on the    /
/ device. /

1398229316({
    / Unique CoSWID ID /    0: "4ca245",
    / tag-version /        12: 23, / Attester-maintained counter /
    / software-name /       1: "Acme R-IoT-OS",
    / software-version /   13: "3.1.4",
    / entity /              2: {
        / entity-name /        31: "Acme Base Attester",
        / role        /        33: 1 / tag-creator /
                            },
    / evidence /            3: {
        / ...file /             17: [
                                    {
            / ...fs-name /              24: "acme_r_iot_os.exe",
            / ...size    /              20: 4502345,
            / ...hash    /               7: [
                                             1, / SHA-256 /
                                             h'05f6b327c173b419
                                               2bd2c3ec248a2922
                                               15eab456611bf7a7
                                               83e25c1782479905'
                                           ]
                                    },
                                    {
            / ...fs-name /              24: "resources.rsc",
            / ...size    /              20: 800945,
            / ...hash    /               7: [
                                              1, / SHA-256 /
                                             h'c142b9aba4280c4b
                                               b8c75f716a43c995
                                               26694caabe529571
                                               f5569bb7dc542f98'
                                           ]
                                    },
                                    {
            / ...fs-name /              24: "common.lib",
            / ...size    /              20: 2309435,
            / ...hash    /               7: [
                                             1, / SHA-256 /
                                             h'a6a9dcdfb3884da5
                                               f884e4e1e8e86299
                                               58c2dbc702741443
                                               a913e34de9333be6'
                                           ]
                                    }
                                ]
                            }
})
```

A.1.6. Attestation Results in JSON format

This is a JSON-format payload that might be the output of a verifier that evaluated the IoT Attestation example immediately above.

This particular verifier knows enough about the TEE attester to be able to pass claims like debug status directly through to the relying party. The verifier also knows the reference values for the measured software components and is able to check them. It informs the relying party that they were correct in the "measres" claim. "Trustus Verifications" is the name of the services that verifies the software component measurements.

```
{
    "eat_nonce" : "jkd8KL-8=Qlzg4",
    "secboot" :  true,
    "dbgstat" :  "disabled-since-boot",
    "oemid" :    "iUWt",
    "ueid" :     "AZj1Ck_2wFhhyIYNE6Y4",
    "swname" :   "Acme R-IoT-OS",
    "swversion" : [
            "3.1.4"
        ],
    "measres" : [
        [
            "Trustus Measurements",
            [
                [ "all" , "success" ]
            ]
       ]
    ]
}
```

A.1.7. JSON-encoded Token with Sumodules

```
{
    "eat_nonce": "lI-IYNE6Rj6O",
    "ueid":     "AJj1Ck_2wFhhyIYNE6Y46g==",
    "secboot":  true,
    "dbgstat":  "disabled-permanently",
    "iat":      1526542894,
    "submods": {
        "Android App Foo" :  {
            "swname": "Foo.app"
        },

        "Secure Element Eat" : [
            "CBOR",

"2D3ShEOhASagWGaoCkiUj4hg0TpGPhkBAFABmPUKT_bAWGHIhg0TpjjqGQECGfryGQEFBBkBBvUZ
AQcDGQEEgmMzLjEBGQEKoWNURUWCL1gg5c-V_ST6txRGdC3VjUPa4XjlX-
K5QpGpKRCC_8JjWgtYQPaQywOIZ3-mJKN3X9fLxOhAnsmBa-MvpHRzOw-
Ywn-67bvJljuctezAPD41s6_At7NbSV3qwJlxIuqGfwe41es="
        ],

        "Linux Android": {
            "swname": "Android"
        },

        "Subsystem J": [
            "JWT",

"eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJpc3MiOiJKLUF0dGVzdGVyIiwiaWF0IjoxNjU
xNzc0ODY4LCJleHAiOm51bGwsImF1ZCI6IiIsInN1YiI6IiJ9.gjw4nFMhLpJUuPXvMPzK1GMjhyJ
q2vWXg1416XKszwQ"
        ]
    }
}
```

A.2. Full Token Examples

A.2.1. Basic CWT Example

This is a simple ECDSA signed CWT-format token.

```
/ This is a full CWT-format token with a very simple payloal. /
/ The main structure visible here is that of the COSE_Sign1.  /

61( 18( [
    h'A10126',                          / protected headers  /
    {},                             / empty unprotected headers /
    h'A20B46024A6B0978DE0A490001020304050607080',    / payload /
    h'9B9B2F5E470000F6A20C8A4157B5763FC45BE759
      9A5334028517768C21AFFB845A56AB557E0C8973
      A07417391243A79C478562D285612E292C622162
      AB233787'                                    / signature /
] ) )
```

A.2.2. Detached EAT Bundle

In this detached EAT bundle, the main token is produced by a HW attestation block. The detached Claims-Set is produced by a TEE and is largely identical to the Simple TEE examples above. The TEE digests its Claims-Set and feeds that digest to the HW block.

In a better example the attestation produced by the HW block would be a CWT and thus signed and secured by the HW block. Since the signature covers the digest from the TEE that Claims-Set is also secured.

The detached EAT bundle itself can be assembled by untrusted software.

```
/ This is a detached EAT bundle tag.  /
/ Note that 602, the tag identifying a detached EAT bundle is not yet
registered with IANA /

602([

    / First part is a full EAT token with claims like nonce and /
    / UEID. Most importantly, it includes a submodule that is a /
    / detached digest which is the hash of the "TEE" claims set /
    / in the next section. The COSE payload follows:            /
    / { /
    /      10: h'948F8860D13A463E', /
    /     256: h'0198F50A4FF6C05861C8860D13A638EA', /
    /     258: 64242, /
    /     262: true, /
    /     263: 3, /
    /     260: ["3.1", 1], /
    /     266: { /
    /         "TEE": [ /
    /             -16, /
    /              h'8DEF652F47000710D9F466A4C666E209  /
    /                DD74F927A1CEA352B03143E188838ABE' /
    /         ] /
    /     } /
    /   }  /
    h'D83DD28443A10126A05866A80A48948F8860D13A463E1901
      00500198F50A4FF6C05861C8860D13A638EA19010219FAF2
      19010504190106F5190107031901048263332E310119010A
      A163544545822F58208DEF652F47000710D9F466A4C666E2
      09DD74F927A1CEA352B03143E188838ABE5840F690CB0388
      677FA624A3775FD7CBC4E8409EC9816BE32FA474733B0F98
      C27FBAEDBBC9963B9CB5ECC03C3E35B3AFC0B7B35B495DEA
      C0997122EA867F07B8D5EB',
    {
       / A CBOR-encoded byte-string wrapped EAT claims-set. It /
       / contains claims suitable for a TEE                   /
       "TEE" : h'a40a48948f8860d13a463e190106f519010702
                 190111818218795858a60064336132340c0101
                 6b41636d6520544545204f530d65332e312e34
                 0282a2181f6b41636d6520544545204f531821
                 01a2181f6b41636d6520544545204f53182102
                 06a111a118186e61636d655f7465655f332e65
                 7865'
    }
 ])
```

```
/ This example contains submodule that is a detached digest,   /
/ which is the hash of a Claims-Set convey outside this token. /
/ Other than that is is the other example of a token from an   /
/ attestation HW block                                         /

{
    / eat_nonce /       10: h'948f8860d13a463e',
    / ueid /           256: h'0198f50a4ff6c05861c8860d13a638ea',
    / oemid /          258: 64242, / Private Enterprise Number /
    / secboot /        262: true,
    / dbgstat /        263: 3, / disabled-permanently /
    / hwversion /      260: [ "3.1", 1 ], / multipartnumeric /
    / submods/         266: {
                                "TEE": [ / detached digest submod /
                                    -16, / SHA-256 /
                                    h'e5cf95fd24fab7144674
                                      2dd58d43dae178e55fe2
                                      b94291a9291082ffc2635
                                      a0b'
                                ]
                            }
}
```

A.2.3. JSON-encoded Detached EAT Bundle

In this bundle there are two detached Claims-Sets, "CS1" and "CS2". The JWT at the start of the bundle has detached signature submodules with hashes of "CS1" and "CS2". TODO: make the JWT actually be correct verifiable JWT.

```
[
    [ "JWT",

"eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJpc3MiOiJKLUF0dGVzdGVyIiwiaWF0IjoxNjU
xNzc0ODY4LCJleHAiOm51bGwsImF1ZCI6IiIsInN1YiI6IiJ9.gjw4nFMhLpJUuPXvMPzK1GMjhyJ
q2vWXg1416XKszwQ"
    ],
    {
        "Audio Subsystem Claims":
"ewogICAgICAgICAgICAibm9uY2UiOiAgICAgImxJK0lZTkU2Umo2TyIsCiAgICAgICAgICAgICJp
YXQiOiAgICAgIDE1MjY1NDI4OTQKICAgICAgICAgfQo=",
        "Graphics Subsystem Claims":
"ewogICAgICAgICAgICAibm9uY2UiOiAgICJsSStJWU5FNlJqNk8iLAogICAgICAgICAgICAiaWF0
IjogICAgIDE1MjY1NDI4OTQKICAgICAgICB9"
    }
]
```

Appendix B. UEID Design Rationale

B.1. Collision Probability

This calculation is to determine the probability of a collision of UEIDs given the total possible entity population and the number of entities in a particular entity management database.

Three different sized databases are considered. The number of devices per person roughly models non-personal devices such as traffic lights, devices in stores they shop in, facilities they work in and so on, even considering individual light bulbs. A device may have individually attested subsystems, for example parts of a car or a mobile phone. It is assumed that the largest database will have at most 10% of the world's population of devices. Note that databases that handle more than a trillion records exist today.

The trillion-record database size models an easy-to-imagine reality over the next decades. The quadrillion-record database is roughly at the limit of what is imaginable and should probably be accommodated. The 100 quadrillion datadbase is highly speculative perhaps involving nanorobots for every person, livestock animal and domesticated bird. It is included to round out the analysis.

Note that the items counted here certainly do not have IP address and are not individually connected to the network. They may be connected to internal buses, via serial links, Bluetooth and so on. This is not the same problem as sizing IP addresses.

People	Devices / Person	Subsystems / Device	Database Portion	Database Size
10 billion	100	10	10%	trillion (10^12)
10 billion	100,000	10	10%	quadrillion (10^15)
100 billion	1,000,000	10	10%	100 quadrillion (10^17)

Table 5

This is conceptually similar to the Birthday Problem where m is the number of possible birthdays, always 365, and k is the number of people. It is also conceptually similar to the Birthday Attack where collisions of the output of hash functions are considered.

The proper formula for the collision calculation is

```
p = 1 - e^{-k^2/(2n)}

p   Collision Probability
n   Total possible population
k   Actual population
```

However, for the very large values involved here, this formula requires floating point precision higher than commonly available in calculators and software so this simple approximation is used. See [BirthdayAttack].

```
p = k^2 / 2n
```

For this calculation:

```
p  Collision Probability
n  Total population based on number of bits in UEID
k  Population in a database
```

Database Size	128-bit UEID	192-bit UEID	256-bit UEID
trillion (10^12)	2 * 10^-15	8 * 10^-35	5 * 10^-55
quadrillion (10^15)	2 * 10^-09	8 * 10^-29	5 * 10^-49
100 quadrillion (10^17)	2 * 10^-05	8 * 10^-25	5 * 10^-45

Table 6

Next, to calculate the probability of a collision occurring in one year's operation of a database, it is assumed that the database size is in a steady state and that 10% of the database changes per year. For example, a trillion record database would have 100 billion states per year. Each of those states has the above calculated probability of a collision.

This assumption is a worst-case since it assumes that each state of the database is completely independent from the previous state. In reality this is unlikely as state changes will be the addition or deletion of a few records.

The following tables gives the time interval until there is a probability of a collision based on there being one tenth the number of states per year as the number of records in the database.

```
t = 1 / ((k / 10) * p)

t  Time until a collision
p  Collision probability for UEID size
k  Database size
```

Database Size	128-bit UEID	192-bit UEID	256-bit UEID
trillion (10^12)	60,000 years	10^24 years	10^44 years
quadrillion (10^15)	8 seconds	10^14 years	10^34 years
100 quadrillion (10^17)	8 microseconds	10^11 years	10^31 years

Table 7

Clearly, 128 bits is enough for the near future thus the requirement that UEIDs be a minimum of 128 bits.

There is no requirement for 256 bits today as quadrillion-record databases are not expected in the near future and because this time-to-collision calculation is a very worst case. A future update of the standard may increase the requirement to 256 bits, so there is a requirement that implementations be able to receive 256-bit UEIDs.

B.2. No Use of UUID

A UEID is not a UUID [RFC4122] by conscious choice for the following reasons.

UUIDs are limited to 128 bits which may not be enough for some future use cases.

Today, cryptographic-quality random numbers are available from common CPUs and hardware. This hardware was introduced between 2010 and 2015. Operating systems and cryptographic libraries give access to this hardware. Consequently, there is little need for implementations to construct such random values from multiple sources on their own.

Version 4 UUIDs do allow for use of such cryptographic-quality random numbers, but do so by mapping into the overall UUID structure of time and clock values. This structure is of no value here yet adds complexity. It also slightly reduces the number of actual bits with entropy.

The design of UUID accommodates the construction of a unique identifier by combination of several identifiers that separately do not provide sufficient uniqueness. UEID takes the view that this construction is no longer needed, in particular because cryptographic-quality random number generators are readily available. It takes the view that hardware, software and/or manufacturing process implement UEID in a simple and direct way.

Appendix C. EAT Relation to IEEE.802.1AR Secure Device Identity (DevID)

This section describes several distinct ways in which an IEEE IDevID [IEEE.802.1AR] relates to EAT, particularly to UEID and SUEID.

[IEEE.802.1AR] orients around the definition of an implementation called a "DevID Module." It describes how IDevIDs and LDevIDs are stored, protected and accessed using a DevID Module. A particular level of defense against attack that should be achieved to be a DevID is defined. The intent is that IDevIDs and LDevIDs can be used with any network protocol or message format. In these protocols and message formats the DevID secret is used to sign a nonce or similar to prove the association of the DevID certificates with the device.

By contrast, EAT standardize a message format that is sent to a relying party, the very thing that is not defined in [IEEE.802.1AR]. Nor does EAT give details on how keys, data and such are stored protected and accessed. EAT is intended to work with a variety of different on-device implementations ranging from minimal protection of assets to the highest levels of asset protection. It does not define any particular level of defense against attack, instead providing a set of security considerations.

EAT and DevID can be viewed as complimentary when used together or as competing to provide a device identity service.

C.1. DevID Used With EAT

As just described, EAT standardizes a message format and [IEEE.802.1AR] doesn't. Vice versa, EAT doesn't define a an device implementation and DevID does.

Hence, EAT can be the message format that a DevID is used with. The DevID secret becomes the attestation key used to sign EATs. The DevID and its certificate chain become the endorsement sent to the verifier.

In this case, the EAT and the DevID are likely to both provide a device identifier (e.g. a serial number). In the EAT it is the UEID (or SUEID). In the DevID (used as an endorsement), it is a device serial number included in the subject field of the DevID certificate. It is probably a good idea in this use for them to be the same serial number or for the UEID to be a hash of the DevID serial number.

C.2. How EAT Provides an Equivalent Secure Device Identity

The UEID, SUEID and other claims like OEM ID are equivalent to the secure device identity put into the subject field of a DevID certificate. These EAT claims can represent all the same fields and values that can be put in a DevID certificate subject. EAT explicitly and carefully defines a variety of useful claims.

EAT secures the conveyance of these claims by having them signed on the device by the attestation key when the EAT is generated. EAT also signs the nonce that gives freshness at this time. Since these claims are signed for every EAT generated, they can include things that vary over time like GPS location.

DevID secures the device identity fields by having them signed by the manufacturer of the device sign them into a certificate. That certificate is created once during the manufacturing of the device and never changes so the fields cannot change.

So in one case the signing of the identity happens on the device and the other in a manufacturing facility, but in both cases the signing of the nonce that proves the binding to the actual device happens on the device.

While EAT does not specify how the signing keys, signature process and storage of the identity values should be secured against attack, an EAT implementation may have equal defenses against attack. One reason EAT uses CBOR is because it is simple enough that a basic EAT implementation can be constructed entirely in hardware. This allows EAT to be implemented with the strongest defenses possible.

C.3. An X.509 Format EAT

It is possible to define a way to encode EAT claims in an X.509 certificate. For example, the EAT claims might be mapped to X.509 v3 extensions. It is even possible to stuff a whole CBOR-encoded unsigned EAT token into a X.509 certificate.

If that X.509 certificate is an IDevID or LDevID, this becomes another way to use EAT and DevID together.

Note that the DevID must still be used with an authentication protocol that has a nonce or equivalent. The EAT here is not being used as the protocol to interact with the rely party.

C.4. Device Identifier Permanence

In terms of permanence, an IDevID is similar to a UEID in that they do not change over the life of the device. They cease to exist only when the device is destroyed.

An SUEID is similar to an LDevID. They change on device life-cycle events.

[IEEE.802.1AR] describes much of this permanence as resistant to attacks that seek to change the ID. IDevID permanence can be described this way because [IEEE.802.1AR] is oriented around the definition of an implementation with a particular level of defense against attack.

EAT is not defined around a particular implementation and must work on a range of devices that have a range of defenses against attack. EAT thus can't be defined permanence in terms of defense against attack. EAT's definition of permanence is in terms of operations and device lifecycle.

Appendix D. CDDL for CWT and JWT

[RFC8392] was published before CDDL was available and thus is specified in prose, not CDDL. Following is CDDL specifying CWT as it is needed to complete this specification. This CDDL also covers the Claims-Set for JWT.

The COSE-related types in this CDDL are defined in [RFC9052].

This however is NOT a normative or standard definition of CWT or JWT in CDDL. The prose in CWT and JWT remain the normative definition.

```
; This is replicated from draft-ietf-rats-uccs

Claims-Set = {
    * $$Claims-Set-Claims
    * Claim-Label .feature "extended-claims-label" => any
}
Claim-Label = int / text
string-or-uri = text

$$Claims-Set-Claims //= ( iss-claim-label => string-or-uri  )
$$Claims-Set-Claims //= ( sub-claim-label => string-or-uri  )
$$Claims-Set-Claims //= ( aud-claim-label => string-or-uri  )
$$Claims-Set-Claims //= ( exp-claim-label => ~time )
$$Claims-Set-Claims //= ( nbf-claim-label => ~time )
$$Claims-Set-Claims //= ( iat-claim-label => ~time )
$$Claims-Set-Claims //= ( cti-claim-label => bytes )

iss-claim-label = JC<"iss", 1>
sub-claim-label = JC<"sub", 2>
aud-claim-label = JC<"aud", 3>
exp-claim-label = JC<"exp", 4>
nbf-claim-label = JC<"nbf", 5>
iat-claim-label = JC<"iat", 6>
cti-claim-label = CBOR-ONLY<7>  ; jti in JWT: different name and text

JSON-ONLY<J> = J .feature "json"
CBOR-ONLY<C> = C .feature "cbor"

; Be sure to have cddl 0.8.29 or higher for this to work
JC<J,C> = JSON-ONLY<J> / CBOR-ONLY<C>
```

```
; A JWT message is either a JWS or JWE in compact serialization form
; with the payload a Claims-Set. Compact serialization is the
; protected headers, payload and signature, each b64url encoded and
; separated by a ".". This CDDL simply matches top-level syntax of of
; a JWS or JWE since it is not possible to do more in CDDL.

JWT-Message = text .regexp "[A-Za-z0-9_=-]+\.[A-Za-z0-9_=-]+\.[A-Za-z0-9_=-]
+"


; Note that the payload of a JWT is defined in claims-set.cddl. That
; definition is common to CBOR and JSON.
```

```
; This is some CDDL describing a CWT at the top level This is
; not normative. RFC 8392 is the normative definition of CWT.

CWT-Messages = CWT-Tagged-Message / CWT-Untagged-Message

; The payload of the COSE_Message is always a Claims-Set

; The contents of a CWT Tag must always be a COSE tag
CWT-Tagged-Message = #6.61(COSE_Tagged_Message)

; An untagged CWT may be a COSE tag or not
CWT-Untagged-Message = COSE_Messages
```

Appendix E. Claim Characteristics

The following is design guidance for creating new EAT claims, particularly those to be registered with IANA.

Much of this guidance is generic and could also be considered when designing new CWT or JWT claims.

E.1. Interoperability and Relying Party Orientation

It is a broad goal that EATs can be processed by Relying Parties in a general way regardless of the type, manufacturer or technology of the device from which they originate. It is a goal that there be general-purpose verification implementations that can verify tokens for large numbers of use cases with special cases and configurations for different device types. This is a goal of interoperability of the semantics of claims themselves, not just of the signing, encoding and serialization formats.

This is a lofty goal and difficult to achieve broadly requiring careful definition of claims in a technology neutral way. Sometimes it will be difficult to design a claim that can represent the semantics of data from very different device types. However, the goal remains even when difficult.

E.2. Operating System and Technology Neutral

Claims should be defined such that they are not specific to an operating system. They should be applicable to multiple large high-level operating systems from different vendors. They should also be applicable to multiple small embedded operating systems from multiple vendors and everything in between.

Claims should not be defined such that they are specific to a software environment or programming language.

Claims should not be defined such that they are specific to a chip or particular hardware. For example, they should not just be the contents of some HW status register as it is unlikely that the same HW status register with the same bits exists on a chip of a different manufacturer.

The boot and debug state claims in this document are an example of a claim that has been defined in this neutral way.

E.3. Security Level Neutral

Many use cases will have EATs generated by some of the most secure hardware and software that exists. Secure Elements and smart cards are examples of this. However, EAT is intended for use in low-security use cases the same as high-security use case. For example, an app on a mobile device may generate EATs on its own.

Claims should be defined and registered on the basis of whether they are useful and interoperable, not based on security level. In particular, there should be no exclusion of claims because they are just used only in low-security environments.

E.4. Reuse of Extant Data Formats

Where possible, claims should use already standardized data items, identifiers and formats. This takes advantage of the expertise put into creating those formats and improves interoperability.

Often extant claims will not be defined in an encoding or serialization format used by EAT. It is preferred to define a CBOR and JSON format for them so that EAT implementations do not require a plethora of encoders and decoders for serialization formats.

In some cases, it may be better to use the encoding and serialization as is. For example, signed X.509 certificates and CRLs can be carried as-is in a byte string. This retains interoperability with the extensive infrastructure for creating and processing X.509 certificates and CRLs.

E.5. Proprietary Claims

EAT allows the definition and use of proprietary claims.

For example, a device manufacturer may generate a token with proprietary claims intended only for verification by a service offered by that device manufacturer. This is a supported use case.

In many cases proprietary claims will be the easiest and most obvious way to proceed, however for better interoperability, use of general standardized claims is preferred.

Appendix F. Endorsements and Verification Keys

The verifier must possess the correct key when it performs the cryptographic part of an EAT verification (e.g., verifying the COSE/JOSE signature). This section describes several ways to identify the verification key. There is not one standard method.

The verification key itself may be a public key, a symmetric key or something complicated in the case of a scheme like Direct Anonymous Attestation (DAA).

RATS Architecture [RATS.Architecture] describes what is called an endorsement. This is an input to the verifier that is usually the basis of the trust placed in an EAT and the attester that generated it. It may contain the public key for verification of the signature on the EAT. It may contain reference values to which EAT claims are compared as part of the verification process. It may contain implied claims, those that are passed on to the relying party in attestation results.

There is not yet any standard format(s) for an endorsement. One format that may be used for an endorsement is an X.509 certificate. Endorsement data like reference values and implied claims can be carried in X.509 v3 extensions. In this use, the public key in the X.509 certificate becomes the verification key, so identification of the endorsement is also identification of the verification key.

The verification key identification and establishment of trust in the EAT and the attester may also be by some other means than an endorsement.

For the components (attester, verifier, relying party,...) of a particular end-end attestation system to reliably interoperate, its definition should specify how the verification key is identified. Usually, this will be in the profile document for a particular attestation system.

F.1. Identification Methods

Following is a list of possible methods of key identification. A specific attestation system may employ any one of these or one not listed here.

The following assumes endorsements are X.509 certificates or equivalent and thus does not mention or define any identifier for endorsements in other formats. If such an endorsement format is created, new identifiers for them will also need to be created.

F.1.1. COSE/JWS Key ID

The COSE standard header parameter for Key ID (kid) may be used. See [RFC9052] and [RFC7515]

COSE leaves the semantics of the key ID open-ended. It could be a record locator in a database, a hash of a public key, an input to a KDF, an authority key identifier (AKI) for an X.509 certificate or other. The profile document should specify what the key ID's semantics are.

F.1.2. JWS and COSE X.509 Header Parameters

COSE X.509 [COSE.X509.Draft] and JSON Web Siganture [RFC7515] define several header parameters (x5t, x5u,...) for referencing or carrying X.509 certificates any of which may be used.

The X.509 certificate may be an endorsement and thus carrying additional input to the verifier. It may be just an X.509 certificate, not an endorsement. The same header parameters are used in both cases. It is up to the attestation system design and the verifier to determine which.

F.1.3. CBOR Certificate COSE Header Parameters

Compressed X.509 and CBOR Native certificates are defined by CBOR Certificates [CBOR.Cert.Draft]. These are semantically compatible with X.509 and therefore can be used as an equivalent to X.509 as described above.

These are identified by their own header parameters (c5t, c5u,...).

F.1.4. Claim-Based Key Identification

For some attestation systems, a claim may be re-used as a key identifier. For example, the UEID uniquely identifies the entity and therefore can work well as a key identifier or endorsement identifier.

This has the advantage that key identification requires no additional bytes in the EAT and makes the EAT smaller.

This has the disadvantage that the unverified EAT must be substantially decoded to obtain the identifier since the identifier is in the COSE/JOSE payload, not in the headers.

F.2. Other Considerations

In all cases there must be some way that the verification key is itself verified or determined to be trustworthy. The key identification itself is never enough. This will always be by some out-of-band mechanism that is not described here. For example, the verifier may be configured with a root certificate or a master key by the verifier system administrator.

Often an X.509 certificate or an endorsement carries more than just the verification key. For example, an X.509 certificate might have key usage constraints and an endorsement might have reference values. When this is the case, the key identifier must be either a protected header or in the payload such that it is cryptographically bound to the EAT. This is in line with the requirements in section 6 on Key Identification in JSON Web Signature [RFC7515].

Appendix G. Changes from Previous Drafts

The following is a list of known changes since the immediately previous drafts. This list is non-authoritative. It is meant to help reviewers see the significant differences. A comprehensive history is available via the IETF Datatracker's record for this document.

G.1. From draft-ietf-rats-eat-14

- Reference to SUIT manifest
- Clarifications about manifest extensibility
- Removed security level claim
- Changed capitalization throughout the document for various terms
- Eliminated use of DEB acronym for detached EAT bundles
- Replicate claim optionality text from CWT and JWT
- Several edits and clarifications for freshness and nonces
- Correct eat_nonce registration for JSON-encoded tokens
- Add security considerations for freshness
- Change/clarify the input to digest algorithm for detached claims sets
- Removed EAN-13 references and IANA registration

- Add section on Claim Trustworthiness to Security Considerations
- Removed section discussing cti/jti and other mention of cti/jti
- Some rework on section 3 including adding back in a **non-normative** reference to UCCS
- Improved wording in section 1.3
- Improvements to abstract
- Appendix C clarifications - say "message" not "protocol"
- Removed "transport security" section from security considerations
- Entirely remove section 4.4 that discussed including keys in claims
- Largely rewrite the first paragraphs in section 1, the introduction
- Mention $$Claims-Set-Claims in prose and require future claims be in CDDL
- Add Carl Wallace as an author

Contributors

Many thanks to the following contributors to draft versions of this document:

Authors' Addresses

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

Giridhar Mandyam
Qualcomm Technologies Inc.
5775 Morehouse Drive
San Diego, California
United States of America
Phone: +1 858 651 7200
Email: mandyam@qti.qualcomm.com

Jeremy O'Donoghue
Qualcomm Technologies Inc.
279 Farnborough Road
Farnborough
GU14 7LS
United Kingdom
Phone: +44 1252 363189
Email: jodonogh@qti.qualcomm.com

Carl Wallace
Red Hound Software, Inc.
Email: carl@redhoundsoftware.com