RATS Working Group
Internet-Draft
Intended status: Standards Track
Expires: November 21, 2022

L. Lundblade
Security Theory LLC
G. Mandyam
J. O'Donoghue
Qualcomm Technologies Inc.

May 20, 2022

The Entity Attestation Token (EAT)
draft-ietf-rats-eat-13

Abstract

An Entity Attestation Token (EAT) provides an attested claims set that describes state and characteristics of an entity, a device like a phone, 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. To a large degree, all this document does is extend CWT and JWT.

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 November 21, 2022.

Copyright Notice

Table of Contents

1. Introduction

EAT provides the definition of a base set of claims that can be made about an entity, a device, some software and/or some hardware. This claims set is received by a relying party who uses it to decide if and how it will interact with the remote entity. It may choose to not trust the entity and not interact with it. It may choose to trust it. It may partially trust it, for example allowing monetary transactions only up to a limit.

EAT defines the encoding of the claims set in CBOR [RFC8949] and JSON [RFC7159]. EAT is an extension to CBOR Web Token (CWT) [RFC8392] and JSON Web Token (JWT) [RFC7519].

The claims set is secured in transit with the same mechanisms used by CWT and JWT, in particular CBOR Object Signing and Encryption (COSE) [RFC8152] and JSON Object Signing and Encryption (JOSE) [RFC7515] [RFC7516]. Authenticity and integrity protection must always be provided. Privacy (encryption) may additionally be provided. The key material used to sign and encrypt is specifically created and provisioned for the purpose of attestation. It is the use of this key material that make the claims set "attested" rather than just some parameters sent to the relying party by the device.

EAT is focused on authenticating, identifying and characterizing implementations where implementations are devices, chips, hardware, software and such. This is distinct from protocols like TLS [RFC8446] that authenticate and identify servers and services. It is equally distinct from protocols like SASL [RFC4422] that authenticate and identify persons.

The notion of attestation is large, ranging over a broad variety of use cases and security levels. Here are a few examples of claims:

- 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 also supports nesting of sets of claims and EAT tokens for use with complex composite devices.

This document uses the terminology and main operational model defined in [RATS.Architecture]. In particular, it can be used for RATS Attestation Evidence and Attestation Results.

1.1. Entity Overview

The document uses the term "entity" to refer to the target of the attestation token. The claims defined in this document are claims about an entity.

An entity is an implementation in hardware, software or both.

An entity is the same as the Attester Target Environment defined in RATS Architecture.

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. Here "entity" never refers to a person or organization.

An entity is never a server or a service.

An entity may be the whole device or it may be a subsystem, a subsystem of a subsystem and so on. EAT allows claims to be organized into submodules, nested EATs and so on. See Section 4.2.19. 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.

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 like 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. CWT, JWT and DEB

An EAT is primarily a claims set about an entity based on one of the following:

- CBOR Web Token (CWT) [RFC8392]
- JSON Web Token (JWT) [RFC7519]

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

This specification extends those specifications by defining additional claims for attestation. This specification also describes the notion of a "profile" that can narrow the definition of an EAT,

ensure interoperability and fill in details for specific usage scenarios. This specification also adds some considerations for registration of future EAT-related claims.

The identification of a protocol element as an EAT, whether CBOR or JSON encoded, follows the general conventions used by CWT, JWT. Largely this depends on the protocol carrying the EAT. In some cases it may be by content type (e.g., MIME type). In other cases it may be through use of CBOR tags. There is no fixed mechanism across all use cases.

This specification adds one more top-level token type:

- Detached EAT Bundle (DEB), Section 5

A DEB is structure to hold a collection of detached claims sets and the EAT that separately provides integrity and authenticity protection for them. It can be either CBOR or JSON encoded.

Last, the definition of other token types is allowed. Of particular use may be a token type that provides no authenticity or integrity protection at all for use with transports like TLS that do provide that.

1.3. CDDL, CBOR and JSON

This document defines Concise Binary Object Representation (CBOR) [RFC8949] and Javascript Object Notation (JSON) [RFC7159] encoding for an EAT. All claims in an EAT MUST use the same encoding except where explicitly allowed. It is explicitly allowed for a nested token to be of a different encoding. Some claims explicitly contain objects and messages that may use a different encoding than the enclosing EAT.

This specification uses Concise Data Definition Language (CDDL) [RFC8610] for all definitions. The implementor interprets the CDDL to come to either the CBOR or JSON encoding. In the case of JSON, Appendix E of [RFC8610] is followed. Additional rules are given in Section 8.2.2 where Appendix E is insufficient.

In most cases where the CDDL for CBOR is different than JSON a CDDL Generic named "JC<>" is used. It is described in Appendix D.

The CWT and JWT specifications were authored before CDDL was available and did not use CDDL. 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.

1.4. Operating Model and RATS Architecture

While it is not required that EAT be used with the RATS operational model described in Figure 1 in [RATS.Architecture], or even that it be used for attestation, this document is oriented around that model.

To summarize, an Attester generates Attestation Evidence. Attestation Evidence is a claims set describing various characteristics of an entity. Attestation Evidence also is usually signed by a key that proves the entity and the evidence it produces are authentic. The claims set includes a nonce or some other means to provide freshness. EAT is designed to carry Attestation Evidence. The Attestation Evidence goes to a Verifier where the signature is verified. Some of the claims may also be checked against Reference Values. The Verifier then produces Attestation Results which is also usually a claims set. EAT is also designed to carry Attestation Results. The Attestation Results go to the Relying Party which is the ultimate consumer of the Remote Attestation Procedure. The Relying Party uses the Attestation Results as needed for the use case, perhaps allowing an entity on the network, allowing a financial transaction or such.

Note that sometimes the Verifier and Relying Party are not separate and thus there is no need for a protocol to carry Attestation Results.

1.4.1. Relationship between Attestation Evidence and Attestation Results

Any claim defined in this document or in the IANA CWT or JWT registry may be used in Attestation Evidence or Attestation Results.

The relationship of claims in Attestation Results to Attestation 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 Attestation 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 Attestation 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 Attestation 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 also modify or transform claims it forwards. This may be to implement some 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 a HW OEM ID Claim in the Attestation Results by performing a look up based on a UEID (serial number) it received in Attestation Evidence.

There are no fixed rules for how a Verifier processes Attestation Evidence to produce Attestation Results. What is important is the Relying Party understand what the Verifier does and what its policies are.

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.

CWT/JWT 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 Attestation 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].

Attestation Evidence: A Claims Set generated by an Attester to be appraised by a Verifier. Attestation 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 Attestation 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.

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.

An EAT may be encoded in CBOR or JSON as defined here. While not encouraged, other documents may define EAT encoding in other formats.

EAT as defined here is always integrity and authenticity protected through use of CWT or JWT. Other token formats using other methods of protection may be defined outside this document.

This document also defines the Detatched EAT Bundle Section 5, a bundle of some detached Claims-Sets and CWTs or JWTs that provide protection for the detached Claims-Set.

The following CDDL defines the top-levels of an EAT token as a socket indicating future token formats may be defined. See Appendix D for the CDDL definitions of a CWT and JWT.

Nesting of EATs is allowed and defined in Section 4.2.19.1.2. This nesting includes nesting of a token that is a different format than the enclosing token. 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 /= DEB-Tagged-Message

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

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

$$EAT-JSON-Token-Formats /= JWT-Message
$$EAT-JSON-Token-Formats /= DEB-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.

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

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 8 on encoding, the CDDL groups turn into CBOR map entries and JSON name/value pairs.

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. Nonce Claim (nonce)

All EATs MUST have a nonce to prevent replay attacks.

This claim is either a single byte or text string or an array of byte or text strings. The array is to accommodate multistage EAT

verification and consumption. See the extensive discussion on attestation freshness in Appendix A of RATS Architecture [RATS.Architecture].

A claim named "nonce" is previously defined and registered with IANA for JWT, but MUST not be used in an EAT. It does not support multiple nonces. No previous nonce claim was defined for CWT.

The nonce MUST have 64 bits of entropy as fewer bits are unlikely to be secure. A maximum nonce size is set to limit the memory required for an implementation. All receivers MUST be able to accommodate the maximum size.

In CBOR, the nonce is a byte string and every bit in the byte string contributes to entropy. The minimum size is 8 bytes. The maximum size is 64 bytes.

In JSON the nonce is a text string. It is assumed that the only characters represented by the lower 7 bits will be used so the text string must be one-seventh longer. The minimum size is 10 bytes. 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 Attestation Evidence or occur in Attestation Results. See Section 1.4.1 for discussion on how Attestation Results relate to Attestation Evidence.

4.2.1. Universal Entity ID Claim (ueid)

A UEID 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 10.1.

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. All implementations MUST be able to receive UEIDs that are 33 bytes long (1 type byte and 256 bits). No UEID longer than 33 bytes SHOULD be sent.

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:

o 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.

A Device Identifier URN is registered for UEIDs. See Section 9.3.4.

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

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

4.2.2. Semi-permanent UEIDs (SUEIDs)

An SEUID is of the same format as a UEID, but it MAY change to a different value on device life-cycle events. Examples of these events are change of ownership, factory reset and on-boarding into an IoT device management system. An entity MAY have both a UEID and SUEIDs, neither, one or the other.

There MAY be multiple SUEIDs. Each one has a text string label the purpose of which is to distinguish it from others in the token. The label MAY name the purpose, application or type of the SUEID. Typically, there will be few SUEDs so there is no need for a formal labeling mechanism like a registry. The EAT profile MAY describe how SUEIDs should be labeled. If there is only one SUEID, the claim remains a map and there still must be a label. For example, the label for the SUEID used by FIDO Onboarding Protocol could simply be "FDO".

There are privacy considerations for SUEIDs. See Section 10.1.

A Device Indentifier URN is registered for SUEIDs. See Section 9.3.4.

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

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

4.2.3. Hardware OEM Identification (oemid)

This 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 integer-based company registry called the Private Enterprise Number (PEN) [PEN].

PENs are often used to create an OID. That is not the case here. They are used only as an integer.

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. Hardware Model Claim (hardware-model)

This 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. Hardware Version Claims (hardware-version-claims)

The hardware version 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.

The hardware version can also be given by a 13-digit [EAN-13]. A new CoSWID version scheme is registered with IANA by this document in Section 9.3.3. An EAN-13 is also known as an International Article Number or most commonly as a bar code.

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

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

4.2.6. Software Name Claim

This is a free-form text claim for the name of the software for the entity or submodule. A CoSWID manifest or other type of manifest can be used instead if this claim is to limited to correctly characterize the SW for the entity or submodule.

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

4.2.7. Software Version Claim

This 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. The Security Level Claim (security-level)

This claim characterizes the entity's ability to defend against attacks aimed at capturing the signing key, forging claims and forging EATs.

The intent of this claim is only to give the recipient a rough idea of the security the entity is aiming for. This is via a simple, non-extensible set of three levels.

This takes a broad view of the range of defenses because EAT is targeted at a broad range of use cases. The least secure level involves minimal SW defenses. The most secure level involves specialized hardware to defend against hardware-based attacks.

Only through expansive certification programs like Common Criteria and FIDO certification is it possible to sharply define security levels. Sharp definition of security levels is not possible here because the IETF doesn't define and operate certification programs. It is also not possible here because any sharp definition of security levels would be a document larger than the EAT specification. Thus, this definition takes the view that the security level definition possible here is a simple, modest, rough characterization.

1 - Unrestricted: An entity is categorized as unrestricted when it doesn't meet the criteria for any of the higher levels. This

level does not indicate there is no protection at all, just that the entity doesn't qualify for the higher levels.

2 - Restricted: Entities at this level MUST meet the criteria defined in Section 4 of FIDO Allowed Restricted Operating Environments [FIDO.AROE]. Examples include TEE's and schemes using virtualization-based security. Security at this level is aimed at defending against large-scale network/remote attacks by having a reduced attack surface.

3 - Hardware: Entities at this level are indicating they have some countermeasures to defend against physical or electrical attacks against the entity. Security at this level is aimed at defending against attackers that physically capture the entity to attack it. Examples include TPMs and Secure Elements.

The security level claimed should be for the weakest point in the entity, not the strongest. For example, if attestation key is protected by hardware, but the rest of the attester is in a TEE, the claim must be for restriced.

This set of three is not extensible so this remains a broad interoperable description of security level.

In particular use cases, alternate claims may be defined that give finer grained information than this claim.

See also the DLOAs claim in Section 4.2.15, a claim that specifically provides information about certifications received.

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

security-level-type = unrestricted /
                      restricted /
                      hardware

unrestricted       = JC< "unrestricted",      1>
restricted         = JC< "restricted",        2>
hardware           = JC< "hardware",          3>
```

4.2.9. Secure Boot Claim (secure-boot)

The 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)

4.2.10. Debug Status Claim (debug-status)

This 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 debug level 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.

4.2.10.1. Enabled

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

4.2.10.2. 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.

4.2.10.3. Disabled Since Boot

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

4.2.10.4. 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.

4.2.10.5. 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 >
```

4.2.11. The Location Claim (location)

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 10.2.

```
$$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 >
```

4.2.12. The Uptime Claim (uptime)

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)
```

4.2.13. The Boot Odometer Claim (odometer)

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

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

4.2.14. The Boot Seed Claim (boot-seed)

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

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.

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

4.2.15. The DLOA (Digital Letter of Approval) Claim (dloas)

A DLOA (Digital Letter of Approval) [DLOA] is an XML 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. When this claim is issued by a Verifier, it MUST be because the entity has received the certification in the DLOA.

This claim MAY contain more than one DLOA. If multiple DLOAs are present, it MUST be because the entity received all of the certifications.

DLOA XML 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
]
```

4.2.16. The Software Manifests Claim (manifests)

This 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 CBOR-encoded EATs the manifest, whatever format it is, MUST be placed in a byte string.

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.

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

manifests-type = [+ manifest-format]

manifest-format = [
    content-type:   uint,
    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

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

4.2.17. The Software Evidence Claim (swevidence)

This claim contains descriptions, lists, evidence or measurements of the software that exists on the entity. 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 software manufacturer.

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 4.2.16.

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

swevidence-type = [+ swevidence-format]

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

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

4.2.18. The Measurement Results Claim (measurement-results)

This 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 swevidence claim, {#swevidence}, 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 >
```

4.2.19. Submodules (submods)

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...

4.2.19.1. 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
```

4.2.19.1.1. Submodule Claims-Set

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

The submodule Claims-Set is produced by the same Attester as the surrounding token. It is secured using the same mechanism as the enclosing token (e.g., it is signed by the same attestation key). It roughly corresponds to an Attester Target 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.

This data type for this type of submodule is a map/object. It is identified when decoding by it’s type being a map/object.

4.2.19.1.2. 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.

4.2.19.1.2.1. 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 DEB, 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
```

4.2.19.1.2.2. 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 DEB

"DEB": 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" / "DEB",
   nested-token : JWT-Message /
                  CBOR-Token-Inside-JSON-Token /
                  Detached-EAT-Bundle
]

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

4.2.19.1.3. 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.

The input to the digest is exactly the byte-string wrapped encoded form of the Claims-Set for the submodule. That Claims-Set can

include other submodules including nested tokens and detached digests.

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 integrity protection for the larger Claims Sets will not be as secure as those originating in hardware block, but the key material and hardware-based claims will be. It is possible for the hardware to enforce hardware access control (memory protection) on the digest registers so that some of the larger claims can be more secure. For example, one register may be writable only by the TEE, so the detached claims from the TEE will have TEE-level security.

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

When decoding a CBOR format token the detached digest type is distringuished from the other types by it being an array. In CBOR the 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 "DEB", then the submodule is a nested token. Otherwise it will contain an algorithm identifier and is a detached digest.

A DEB, 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 DEB. 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 = [
   algorithm : JC< text, int >
   digest    : binary-data
]
```

4.2.19.2. 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 the nonce.

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

4.2.19.3. 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.

4.2.19.4. 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 Attestation Evidence or Attestation Results. When these claims appear in Attestation Evidence, they SHOULD not be passed through the Verifier into Attestation Results.

4.3.1. Token ID Claim (cti and jti)

CWT defines the "cti" claim. JWT defines the "jti" claim. These are equivalent to each other in EAT and carry a unique token identifier as they do in JWT and CWT. They may be used to defend against re use of the token but are distinct from the nonce that is used by the Relying Party to guarantee freshness and defend against replay.

4.3.2. Timestamp claim (iat)

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 float-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 epoch time 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.

4.3.3. The Profile Claim (profile)

See Section 7 for the detailed description of a profile.

A profile is identified 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 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 8.2.1 for OID and URI encoding.

Note that this is named "eat_profile" for JWT and is distinct from the already registered "profile" claim in the JWT claims registry.

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

4.3.4. The Intended Use Claim (intended-use)

EAT’s may be used in the context of several different applications. The intended-use 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 intended-use 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 >
```

4.4. Including Keys

An EAT may include a cryptographic key such as a public key. The signing of the EAT binds the key to all the other claims in the token.

The purpose for inclusion of the key may vary by use case. For example, the key may be included as part of an IoT device onboarding protocol. When the FIDO protocol includes a public key in its attestation message, the key represents the binding of a user, device and Relying Party. This document describes how claims containing keys should be defined for the various use cases. It does not define specific claims for specific use cases.

Keys in CBOR format tokens SHOULD be the COSE_Key format [RFC8152] and keys in JSON format tokens SHOULD be the JSON Web Key format [RFC7517]. These two formats support many common key types. Their use avoids the need to decode other serialization formats. These two formats can be extended to support further key types through their IANA registries.

The general confirmation claim format [RFC8747], [RFC7800] may also be used. It provides key encryption. It also allows for inclusion by reference through a key ID. The confirmation claim format may employed in the definition of some new claim for a a particular use case.

When the actual confirmation claim is included in an EAT, this document associates no use case semantics other than proof of possession. Different EAT use cases may choose to associate further semantics. The key in the confirmation claim MUST be protected in the same way as the key used to sign the EAT. That is, the same, equivalent or better hardware defenses, access controls, key generation and such must be used.

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 DEB 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 DEB. 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

DEB is CBOR-encoded, each Claims-Set is wrapped in a byte string. When the DEB is JSON-encoded, each Claims-Set is base64url encoded. All the detached Claims-Sets MUST be encoded in the same format as the DEB. No mixing of encoding formats is allowed for the Claims-Sets in a DEB.

For CBOR-encoded DEBs, 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.

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

DEB-Tagged-Message   = #6.TBD(DEB-Untagged-Message)
DEB-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. 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.

6.1. 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.

6.1.1. COSE/JWS Key ID

The COSE standard header parameter for Key ID (kid) may be used. See [RFC8152] 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.

6.1.2. 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.

6.1.3. 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,...).

6.1.4. 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.

6.2. 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].

7. Profiles

This EAT specification does not gaurantee that implementations of it will interoperate. The variability in this specification is necessary to accommodate the widely varying use cases. An EAT profile narrows the specification for a specific use case. An ideal EAT profile will guarantee interoperability.

The profile can be named in the token using the profile claim described in Section 4.3.3.

A profile can apply to Attestation Evidence or to Attestation Results or both.

7.1. 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.

In some cases CDDL may be created that replaces CDDL in this or other document to express some profile requirements. For example, to require the altitude data item in the location claim, CDDL can be written that replicates the location claim with the altitude no longer optional.

7.2. List of Profile Issues

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

7.2.1. Use of JSON, CBOR or both

The profile should indicate whether the token format should be CBOR, JSON, both or even some other encoding. If some other encoding, a specification for how the CDDL described here is serialized in that encoding is necessary.

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

7.2.2. CBOR Map and Array Encoding

The profile should indicate whether definite-length arrays/maps, indefinite-length arrays/maps or both are allowed. A good default is to allow only definite-length arrays/maps.

An alternate is to allow both definite and indefinite-length arrays/maps. The decoder should accept either. Encoders that need to fit on very small hardware or be actually implement in hardware can use indefinite-length encoding.

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

7.2.3. CBOR String Encoding

The profile should indicate whether definite-length strings, indefinite-length strings or both are allowed. A good default is to allow only definite-length strings. As with map and array encoding, allowing indefinite-length strings can be beneficial for some smaller implementations.

7.2.4. CBOR Preferred Serialization

The profile should indicate whether encoders must use preferred serialization. The profile should indicate whether decoders must accept non-preferred serialization.

7.2.5. 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.

The profile should list the protections that must be supported by all decoders implementing the profile. The encoders them must implement a subset of what is listed for the decoders, perhaps only one.

Implementations may choose to sign or MAC before encryption so that the implementation layer doing the signing or MACing can be the smallest. It is often easier to make smaller implementations more secure, perhaps even implementing in solely in hardware. The key material for a signature or MAC is a private key, while for encryption it is likely to be a public key. The key for encryption requires less protection.

7.2.6. COSE/JOSE Algorithms

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 submodules are used, the COSE algorithms allowed for their digests should also be in the profile.

7.2.7. DEB Support

A Detatched EAT Bundle Section 5 is a special case message that will not often be used. A profile may prohibit its use.

7.2.8. Verification Key Identification

Section Section 6 describes a number of methods for identifying a verification key. The profile document should specify one of these or one that is not described. The ones described in this document are only roughly described. The profile document should go into the full detail.

7.2.9. 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.

7.2.10. Freshness

Just about every use case will require some means of knowing the EAT is recent enough and not a replay of an old token. The profile should describe how freshness is achieved. The section on Freshness in [RATS.Architecture] describes some of the possible solutions to achieve this.

7.2.11. Required Claims

The profile can list claims whose absence results in Verification failure.

7.2.12. Prohibited Claims

The profile can list claims whose presence results in Verification failure.

7.2.13. Additional Claims

The profile may describe entirely new claims. These claims can be required or optional.

7.2.14. Refined Claim Definition

The profile may lock down optional aspects of individual claims. For example, it may require altitude in the location claim, or it may require that HW Versions always be described using EAN-13.

7.2.15. 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 unncessary because the surrounding or carrying protocol identifies the object as an EAT.

7.2.16. Manifests and Software Evidence Claims

The profile should specify which formats are allowed for the manifests and software evidence claims. The profile may also go on to say which parts and options of these formats are used, allowed and prohibited.

8. 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.

8.1. 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.

8.2. Encoding Data Types

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

8.2.1. 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 "[0-9\.]+"

general-uri = JC< text, ~uri >
```

8.2.2. 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.

8.2.3. Labels

Map labels, including Claims-Keys and Claim-Names, and enumerated-type values are always integers when encoding in CBOR and strings when encoding in JSON. There is an exception to this for naming submodules and detached claims sets in a DEB. These are strings in CBOR.

The CDDL in most cases gives both the integer label and the string label as it is not convenient to have conditional CDDL for such.

8.3. CBOR Interoperability

CBOR allows data items to be serialized in more than one form. If the sender uses a form that the receiver can’t decode, there will not be interoperability.

This specification gives no blanket requirements to narrow CBOR serialization for all uses of EAT. This allows individual uses to tailor serialization to the environment. It also may result in EAT implementations that don’t interoperate.

One way to guarantee interoperability is to clearly specify CBOR serialization in a profile document. See Section 7 for a list of serialization issues that should be addressed.

EAT will be commonly used where the entity generating the attestation is constrained and the receiver/Verifier of the attestation is a capacious server. Following is a set of serialization requirements that work well for that use case and are guaranteed to interoperate. Use of this serialization is recommended where possible, but not required. An EAT profile may just reference the following section rather than spell out serialization details.

8.3.1. EAT Constrained Device Serialization

- Preferred serialization described in section 4.1 of [RFC8949] is not required. The EAT decoder must accept all forms of number serialization. The EAT encoder may use any form it wishes.

- The EAT decoder must accept indefinite length arrays and maps as described in section 3.2.2 of [RFC8949]. The EAT encoder may use indefinite length arrays and maps if it wishes.

- The EAT decoder must accept indefinite length strings as described in section 3.2.3 of [RFC8949]. The EAT encoder may use indefinite length strings if it wishes.

- Sorting of maps by key is not required. The EAT decoder must not rely on sorting.

- Deterministic encoding described in Section 4.2 of [RFC8949] is not required.

- Basic validity described in section 5.3.1 of [RFC8949] must be followed. The EAT encoder must not send duplicate map keys/labels or invalid UTF-8 strings.

8.4. Collected CDDL

8.4.1. 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 "[0-9\.]+"

  general-uri = JC< text, ~uri >

  $$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 //=
      ( security-level-label => security-level-type )

  security-level-type = unrestricted /
                        restricted /
                        hardware

  unrestricted       = JC< "unrestricted",      1>
  restricted         = JC< "restricted",        2>
  hardware           = JC< "hardware",          3>

  $$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 //= (odometer-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:   uint,
      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

  suit-directive-process-dependency = 19

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

  swevidence-type = [+ swevidence-format]

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

  $$swevidence-body-cbor /= bytes .cbor untagged-coswid
  $$swevidence-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 = [
     algorithm : JC< text, int >
     digest    : binary-data
  ]


  DEB-Messages = DEB-Tagged-Message / DEB-Untagged-Message

  DEB-Tagged-Message   = #6.TBD(DEB-Untagged-Message)
  DEB-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< "hwvers",      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 >

  security-level-label   = JC< "seclevel",    TBD >
  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 >
  swevidence-label       = JC< "swevidence",  TBD >
  measurement-results-label = JC< "measres" ,  TBD >
  odometer-label         = JC< "odometer",    TBD >
```

8.4.2. 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 /= DEB-Tagged-Message

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

8.4.3. JSON-Specific CDDL

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

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


Nested-Token = JSON-Nested-Token

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

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

9. IANA Considerations

9.1. 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.

9.2. 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.

9.2.1. 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.

9.2.2. 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 SW 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.

9.2.3. 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.

9.2.4. 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.

9.2.5. 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.

9.3. 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 (except for the nonce claim which is already registered for JWT, but not registered for CWT).

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.

9.3.1. 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: "nonce" (already registered for JWT)
- Claim Key: TBD (requested value 10)
- Claim Value Type(s): byte string
- Change Controller: IESG
- Specification Document(s): [OpenIDConnectCore], *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: 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*

9.3.2. To be Assigned Claims

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

- Claim Name: Security Level
- Claim Description: Characterization of the security of an Attester or submodule
- JWT Claim Name: "seclevel"
- Claim Key: TBD
- Claim Value Type(s): integer or string
- Change Controller: IESG
- Specification Document(s): *this document*
- 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: SW Name

- Claim Description: The name of the SW 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: SW Version
- Claim Description: The version of SW 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: SW Manifests
- Claim Description: Manifests describing the SW 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: SW Evidence
- Claim Description: Measurements of the SW, memory configuration and such on the entity
- JWT Claim Name: "swevidence"

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

9.3.3. Version Schemes Registered by this Document

IANA is requested to register a new value in the "Software Tag Version Scheme Values" established by [CoSWID].

The new value is a version scheme a 13-digit European Article Number [EAN-13]. An EAN-13 is also known as an International Article Number or most commonly as a bar code. This version scheme is the ASCII text representation of EAN-13 digits, the same ones often printed with a bar code. This version scheme must comply with the EAN allocation and assignment rules. For example, this requires the manufacturer to obtain a manufacture code from GS1.

Index	Version Scheme Name	Specification
5	ean-13	This document

9.3.4. 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

9.3.5. 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

10. 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.

10.1. 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.

10.2. 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.

10.3. Replay Protection and Privacy

EAT offers 2 primary mechanisms for token replay protection (also sometimes known as token "freshness"): the cti/jti claim and the nonce claim. The cti/jti claim in a CWT/JWT is a field that may be optionally included in the EAT and is in general derived on the same device in which the entity is instantiated. The nonce claim is based on a value that is usually derived remotely (outside of the entity). These claims can be used to extract and convey personally-identifying information either inadvertently or by intention. For instance, an implementor may choose a cti 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 (e.g., if the account name is used to derive the nonce). In order to avoid the conveyance of privacy-related information in

either the cti/jti or nonce claims, these fields 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.

11. 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. In addition, implementors should consider the following.

11.1. 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 manfuacturer 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.

11.1.1. 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.

11.2. Transport Security

As stated in Section 8 of [RFC8392], "The security of the CWT relies upon on the protections offered by COSE". Similar considerations apply to EAT when sent as a CWT. However, EAT introduces the concept of a nonce to protect against replay. Since an EAT may be created by an entity that may not support the same type of transport security as the consumer of the EAT, intermediaries may be required to bridge

communications between the entity and consumer. As a result, it is RECOMMENDED that both the consumer create a nonce, and the entity leverage the nonce along with COSE mechanisms for encryption and/or signing to create the EAT.

Similar considerations apply to the use of EAT as a JWT. Although the security of a JWT leverages the JSON Web Encryption (JWE) and JSON Web Signature (JWS) specifications, it is still recommended to make use of the EAT nonce.

11.3. 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.

12. References

12.1. Normative References

[CoSWID] Birkholz, H., Fitzgerald-McKay, J., Schmidt, C., and D. Waltermire, "Concise Software Identification Tags", draft-ietf-sacm-coswid-21 (work in progress), March 2022.

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

[EAN-13] GS1, "International Article Number - EAN/UPC barcodes", 2019, <https://www.gs1.org/standards/barcodes/ean-upc>.

[FIDO.AROE] The FIDO Alliance, "FIDO Authenticator Allowed Restricted Operating Environments List", November 2020, <https://fidoalliance.org/specs/fido-security-requirements/fido-authenticator-allowed-restricted-operating-environments-list-v1.2-fd-20201102.html>.

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

[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>.

[OpenIDConnectCore] Sakimura, N., Bradley, J., Jones, M., Medeiros, B. D., and C. Mortimore, "OpenID Connect Core 1.0 incorporating errata set 1", November 2014, <https://openid.net/specs/openid-connect-core-1_0.html>.

[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>.

[RFC7159] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", RFC 7159, DOI 10.17487/RFC7159, March 2014, <https://www.rfc-editor.org/info/rfc7159>.

[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>.

[RFC7516] Jones, M. and J. Hildebrand, "JSON Web Encryption (JWE)", RFC 7516, DOI 10.17487/RFC7516, May 2015, <https://www.rfc-editor.org/info/rfc7516>.

[RFC7517] Jones, M., "JSON Web Key (JWK)", RFC 7517, DOI 10.17487/RFC7517, May 2015, <https://www.rfc-editor.org/info/rfc7517>.

[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>.

[RFC7800] Jones, M., Bradley, J., and H. Tschofenig, "Proof-of-Possession Key Semantics for JSON Web Tokens (JWTs)", RFC 7800, DOI 10.17487/RFC7800, April 2016, <https://www.rfc-editor.org/info/rfc7800>.

[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>.

[RFC8152] Schaad, J., "CBOR Object Signing and Encryption (COSE)", RFC 8152, DOI 10.17487/RFC8152, July 2017, <https://www.rfc-editor.org/info/rfc8152>.

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

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

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

[RFC8747] Jones, M., Seitz, L., Selander, G., Erdtman, S., and H. Tschofenig, "Proof-of-Possession Key Semantics for CBOR Web Tokens (CWTs)", RFC 8747, DOI 10.17487/RFC8747, March 2020, <https://www.rfc-editor.org/info/rfc8747>.

[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>.

[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>.

[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", July 2014, <https://earth-info.nga.mil/php/download.php?file=coord-wgs84>.

12.2. Informative References

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

[CBOR.Cert.Draft] Mattsson, J. P., Selander, G., Raza, S., Hoeglund, J., and M. Furuhed, "CBOR Encoded X.509 Certificates (C509 Certificates)", draft-ietf-cose-cbor-encoded-cert-03 (work in progress), January 2022.

[Common.Criteria] "Common Criteria for Information Technology Security Evaluation", April 2017, <https://www.commoncriteriaportal.org/cc/>.

[COSE.X509.Draft]
Schaad, J., "CBOR Object Signing and Encryption (COSE): Header parameters for carrying and referencing X.509 certificates", draft-ietf-cose-x509-08 (work in progress), December 2020.

[FIPS-140]
National Institue of Standards, "Security Requirements for Cryptographic Modules", May 2001, <https://csrc.nist.gov/publications/detail/fips/140/2/final>.

[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", draft-ietf-rats-architecture-15 (work in progress), February 2022.

[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>.

[RFC4422] Melnikov, A., Ed. and K. Zeilenga, Ed., "Simple Authentication and Security Layer (SASL)", RFC 4422, DOI 10.17487/RFC4422, June 2006, <https://www.rfc-editor.org/info/rfc4422>.

[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>.

[RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <https://www.rfc-editor.org/info/rfc8446>.

[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>.

[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, DEB 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. /

   {
       / nonce /           10: h'948f8860d13a463e',
       / security-level / 261: 2, / restricted /
       / secure-boot /    262: true,
       / debug-status /   263: 2, / disabled-since-boot /
       / manfests /       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.         /

{
    / nonce /           10: h'948f8860d13a463e8e',
    / UEID /           256: h'0198f50a4ff6c05861c8860d13a638ea',
    / HW OEM ID /      258: h'894823', / IEEE OUI format OEM ID /
    / HW Model ID /    259: h'549dcecc8b987c737b44e40f7c635ce8'
                              / Hash of chip model name /,
    / HW Version /     260: ["1.3.4", 1], / Multipartnumeric version /
    / SW Name /        271: "Acme OS",
    / SW Version /     272: ["3.5.5", 1],
    / secure-boot /    262: true,
    / debug-status /   263: 3, / permanent-disable  /
    / timestamp (iat) /  6: 1526542894,
    / security-level / 261: 2, / restricted OS /
    / submods / 266: {
        / A submodule to hold some claims about the circuit board /
        "board" :  {
            / HW OEM ID /   258: h'9bef8787eba13e2c8f6e7cb4b1f4619a',
            / HW Model ID / 259: h'ee80f5a66c1fb9742999a8fdab930893'
                                    / Hash of board module name /,
            / HW Version /  260: ["2.0a", 2] / multipartnumeric+suffix /
        },

        / A submodule to hold claims about the overall device /
        "device" :  {
            / HW OEM ID /   258: 61234, / PEN Format OEM ID /
            / HW Version /  260: ["4012345123456", 5] / EAN-13 format (barco
de) /
        }
    }
}
```

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). /

   {
       / nonce /           10: h'948f8860d13a463e',
       / UEID /           256: h'0198f50a4ff6c05861c8860d13a638ea',
       / OEMID /          258: 64242, / Private Enterprise Number /
       / security-level / 261: 3, / hardware level security /
       / secure-boot /    262: true,
       / debug-status /   263: 3, / disabled-permanently /
       / HW version /     260: [ "3.1", 1 ] / Type is multipartnumeric /
   }
```

A.1.4. Key / Key Store Attestation

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

   {
       / nonce /           10: h'948f8860d13a463e',
       / security-level / 261: 2, / restricted /
       / secure-boot /    262: true,
       / debug-status /   263: 2, / disabled-since-boot /
       / manfests /       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.1.5. 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.         /

{
    / nonce /           10: h'948f8860d13a463e8e',
    / UEID /           256: h'0198f50a4ff6c05861c8860d13a638ea',
    / HW OEM ID /      258: h'894823', / IEEE OUI format OEM ID /
    / HW Model ID /    259: h'549dcecc8b987c737b44e40f7c635ce8'
                              / Hash of chip model name /,
    / HW Version /     260: ["1.3.4", 1], / Multipartnumeric version /
    / SW Name /        271: "Acme OS",
    / SW Version /     272: ["3.5.5", 1],
    / secure-boot /    262: true,
    / debug-status /   263: 3, / permanent-disable  /
    / timestamp (iat) /  6: 1526542894,
    / security-level / 261: 2, / restricted OS /
    / submods / 266: {
        / A submodule to hold some claims about the circuit board /
        "board" :  {
            / HW OEM ID /   258: h'9bef8787eba13e2c8f6e7cb4b1f4619a',
            / HW Model ID / 259: h'ee80f5a66c1fb9742999a8fdab930893'
                                    / Hash of board module name /,
            / HW Version /  260: ["2.0a", 2] / multipartnumeric+suffix /
        },

        / A submodule to hold claims about the overall device /
        "device" :  {
            / HW OEM ID /   258: 61234, / PEN Format OEM ID /
            / HW Version /  260: ["4012345123456", 5] / EAN-13 format (barco
de) /
        }
    }
}
```

A.1.6. 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). /

   {
       / nonce /           10: h'948f8860d13a463e',
       / UEID /           256: h'0198f50a4ff6c05861c8860d13a638ea',
       / OEMID /          258: 64242, / Private Enterprise Number /
       / security-level / 261: 3, / hardware level security /
       / secure-boot /    262: true,
       / debug-status /   263: 3, / disabled-permanently /
       / HW version /     260: [ "3.1", 1 ] / Type is multipartnumeric /
   }
```

A.1.7. 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.                  /


   {
       / nonce /           10: h'948f8860d13a463e',
       / security-level / 261: 2, / restricted /
       / secure-boot /    262: true,
       / debug-status /   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"   /
                              ],
      / expiration /       4: 1634324274, / 2021-10-15T18:57:54Z /
      / creation time /    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 /
           / nonce /              10: h'948f8860d13a463e',
             / security-level / 261: 1, / unrestricted /
             / secure-boot /    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.1.8. SW 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.            /

  {
      / nonce /           10: h'948f8860d13a463e',
      / security-level / 261: 2, / restricted /
      / secure-boot /    262: true,
      / debug-status /   263: 2, / disabled-since-boot /
      / OEMID /          258: h'8945ad', / IEEE CID based /
      / UEID /           256: h'0198f50a4ff6c05861c8860d13a638ea',
      / sumods /         266: {
                              "OS" : {
          / security-level /     261: 2, / restricted /
          / secure-boot /        262: true,
          / debug-status /       263: 2, / disabled-since-boot /
          / swevidence /         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
```

```
                                    6d6520426173652041747465737465
                                    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.1.9. 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 security level directly through to the Relying Party. The Verifier also knows the Reference Values for the measured SW components and is able to check them. It informs the Relying Party that they were correct in the swresults claim. "Trustus Verifications" is the name of the services that verifies the SW component measurements.

```
{
    "eat_nonce" : "jkd8KL-8=Qlzg4",
    "seclevel" : "restricted",
    "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.1.10. JSON-encoded Token with Sumodules

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

        "Secure Element Eat" : [
            "CBOR",
            "2D3ShEOhASagWGaoCkiUj4hg0TpGPhkBAFABmPUKT_bAWGHIhg0TpjjqGQECGfr
yGQEFBBkBBvUZAQcDGQEEgmMzLjEBGQEKoWNURUWCL1gg5c-V_ST6txRGdC3VjUPa4XjlX-K5QpG
pKRCC_8JjWgtYQPaQywOIZ3-mJKN3X9fLxOhAnsmBa-MvpHRzOw-Ywn-67bvJljuctezAPD41s6_
At7NbSV3qwJlxIuqGfwe41es="
        ],

        "Linux Android": {
            "seclevel": "unrestricted"
        },

        "Subsystem J": [
            "JWT",
            "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJpc3MiOiJKLUF0dGVzdGVyIi
wiaWF0IjoxNjUxNzc0ODY4LCJleHAiOm51bGwsImF1ZCI6IiIsInN1YiI6IiJ9.gjw4nFMhLpJUu
PXvMPzK1GMjhyJq2vWXg1416XKszwQ"
        ]
    }
}
```

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'A20B46024A6B0978DE0A4900010203040506070 8',    / payload /
    h'9B9B2F5E470000F6A20C8A4157B5763FC45BE759
      9A5334028517768C21AFFB845A56AB557E0C8973
      A07417391243A79C478562D285612E292C622162
      AB233787'                                   / signature /
] ) )
```

A.2.2. Detached EAT Bundle

In this DEB 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 DEB itself can be assembled by untrusted SW.

```
/ This is a detached EAT bundle (DEB) tag.  /
/ Note that 602, the tag identifying a DEB 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, /
    /     261: 4,  /
    /     262: true, /
    /     263: 3, /
    /     260: ["3.1", 1], /
    /     266: { /
    /         "TEE": [ /
    /             -16, /
    /              h'E5CF95FD24FAB71446742DD58D43DAE1 /
    /                78E55FE2B94291A9291082FFC2635A0B' /
    /         ] /
    /     } /
    /   }  /
    h'D83DD28443A10126A05866A80A48948F8860D13A463E1901
      00500198F50A4FF6C05861C8860D13A638EA19010219FAF2
      19010504190106F5190107031901048263332E310119010A
      A163544545822F5820E5CF95FD24FAB71446742DD58D43DA
      E178E55FE2B94291A9291082FFC2635A0B5840F690CB0388
      677FA624A3775FD7CBC4E8409EC9816BE32FA474733B0F98
      C27FBAEDBBC9963B9CB5ECC03C3E35B3AFC0B7B35B495DEA
      C0997122EA867F07B8D5EB',
    {
       / A CBOR-encoded byte-string wrapped EAT claims-set. It /
       / contains claims suitable for a TEE                   /
       "TEE" : h'a50a48948f8860d13a463e19010503190106
                 f5190107021901118218218795858a6006433
                 6132340c01016b41636d6520544545204f53
                 0d65332e312e340282a2181f6b41636d6520
                 544545204f53182101a2181f6b41636d6520
                 544545204f5318210206a111a118186e6163
                 6d655f7465655f332e657865'
    }
 ])
```

```
   / 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                                         /

   {
       / nonce /           10: h'948f8860d13a463e',
       / UEID /           256: h'0198f50a4ff6c05861c8860d13a638ea',
       / OEMID /          258: 64242, / Private Enterprise Number /
       / security-level / 261: 3, / hardware level security /
       / secure-boot /    262: true,
       / debug-status /   263: 3, / disabled-permanently /
       / hw version /     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.eyJpc3MiOiJKLUF0dGVzdGVyIiwiaWF0
IjoxNjUxNzc0ODY4LCJleHAiOm51bGwsImF1ZCI6IiIsInN1YiI6IiJ9.gjw4nFMhLpJUuPXvMPz
K1GMjhyJq2vWXg1416XKszwQ"
    ],
    {
        "Audio Subsystem Claims": "ewogICAgICAgICAgICAibm9uY2UiOiAgICAgImxJK
0lZTkU2Umo2TyIsCiAgICAgICAgICAgICJpYXQiOiAgICAgIDE1MjY1NDI4OTQKICAgICAgICAgf
Qo=",
        "Graphics Subsystem Claims": "ewogICAgICAgICAgICAibm9uY2UiOiAgICJsSS
tJWU5FNlJqNk8iLAogICAgICAgICAgICAiaWF0IjogICAgIDE1MjY1NDI4OTQKICAgICAgICB9"
    }
]
```

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)

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 SW 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

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

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.

UUIDs seem to have been designed for scenarios where the implementor does not have full control over the environment and uniqueness has to be constructed from identifiers at hand. UEID takes the view that

hardware, software and/or manufacturing process directly implement UEID in a simple and direct way. It takes the view that cryptographic quality random number generators are readily available as they are implemented in commonly used CPU hardware.

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 are used with an open set of network protocols for authentication and such. In these protocols the DevID secret is used to sign a nonce or similar to proof the association of the DevID certificates with the device.

By contrast, EAT defines network protocol for proving trustworthiness to a Relying Party, the very thing that is not defined in [IEEE.802.1AR]. Nor does not 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 defines a network protocol and [IEEE.802.1AR] doesn’t. Vice versa, EAT doesn’t define a an device implementation and DevID does.

Hence, EAT can be the network protocol 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.

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. Changes from Previous Drafts

The following is a list of known changes from the previous drafts. This list is non-authoritative. It is meant to help reviewers see the significant differences.

E.1. From draft-rats-eat-01

- Added UEID design rationale appendix

E.2. From draft-mandyam-rats-eat-00

This is a fairly large change in the orientation of the document, but no new claims have been added.

- Separate information and data model using CDDL.
- Say an EAT is a CWT or JWT
- Use a map to structure the boot_state and location claims

E.3. From draft-ietf-rats-eat-01

- Clarifications and corrections for OEMID claim
- Minor spelling and other fixes
- Add the nonce claim, clarify jti claim

E.4. From draft-ietf-rats-eat-02

- Roll all EUIs back into one UEID type
- UEIDs can be one of three lengths, 128, 192 and 256.

- Added appendix justifying UEID design and size.
- Submods part now includes nested eat tokens so they can be named and there can be more tha one of them
- Lots of fixes to the CDDL
- Added security considerations

E.5. From draft-ietf-rats-eat-03

- Split boot_state into secure-boot and debug-disable claims
- Debug disable is an enumerated type rather than Booleans

E.6. From draft-ietf-rats-eat-04

- Change IMEI-based UEIDs to be encoded as a 14-byte string
- CDDL cleaned up some more
- CDDL allows for JWTs and UCCSs
- CWT format submodules are byte string wrapped
- Allows for JWT nested in CWT and vice versa
- Allows UCCS (unsigned CWTs) and JWT unsecured tokens
- Clarify tag usage when nesting tokens
- Add section on key inclusion
- Add hardware version claims
- Collected CDDL is now filled in. Other CDDL corrections.
- Rename debug-disable to debug-status; clarify that it is not extensible
- Security level claim is not extensible
- Improve specification of location claim and added a location privacy section
- Add intended use claim

E.7. From draft-ietf-rats-eat-05

- CDDL format issues resolved
- Corrected reference to Location Privacy section

E.8. From draft-ietf-rats-eat-06

- Added boot-seed claim
- Rework CBOR interoperability section
- Added profiles claim and section

E.9. From draft-ietf-rats-eat-07

- Filled in IANA and other sections for possible preassignment of Claim Keys for well understood claims

E.10. From draft-ietf-rats-eat-08

- Change profile claim to be either a URL or an OID rather than a test string

E.11. From draft-ietf-rats-eat-09

- Add SUEIDs
- Add appendix comparing IDevID to EAT
- Added section on use for Evidence and Attestation Results
- Fill in the key ID and endorsements identificaiton section
- Remove origination claim as it is replaced by key IDs and endorsements
- Added manifests and software evidence claims
- Add string labels non-claim labels for use with JSON (e.g. labels for members of location claim)
- EAN-13 HW versions are no longer a separate claim. Now they are folded in as a CoSWID version scheme.

E.12. From draft-ietf-rats-eat-10

- Hardware version is made into an array of two rather than two claims
- Corrections and wording improvements for security levels claim
- Add swresults claim
- Add dloas claim - Digitial Letter of Approvals, a list of certifications
- CDDL for each claim no longer in a separate sub section
- Consistent use of terminology from RATS architecture document
- Consistent use of terminology from CWT and JWT documents
- Remove operating model and procedures; refer to CWT, JWT and RATS architecture instead
- Some reorganization of Section 1
- Moved a few references, including RATS Architecture, to informative.
- Add detached submodule digests and detached eat bundles (DEBs)
- New simpler and more universal scheme for identifying the encoding of a nested token
- Made clear that CBOR and JSON are only mixed when nesting a token in another token
- Clearly separate CDDL for JSON and CBOR-specific data items
- Define UJCS (unsigned JWTs)
- Add CDDL for a general Claims-Set used by UCCS, UJCS, CWT, JWT and EAT
- Top level CDDL for CWT correctly refers to COSE
- OEM ID is specifically for HW, not for SW
- HW OEM ID can now be a PEN
- HW OEM ID can now be a 128-bit random number

- Expand the examples section
- Add software and version claims as easy / JSON alternative to CoSWID

E.13. From draft-ietf-rats-eat-11

- Add HW model claim
- Change reference for CBOR OID draft to RFC 9090
- Correct the iat claim in some examples
- Make HW Version just one claim rather than 3 (device, board and chip)
- Remove CDDL comments from CDDL blocks
- More clearly define "entity" and use it more broadly, particularly instead of "device"
- Re do early allocation of CBOR labels since last one didn’t complete correctly
- Lots of rewording and tightening up of section 1
- Lots of wording improvements in section 3, particularly better use of normative language
- Improve wording in submodules section, particularly how to distinguish types when decoding
- Remove security-level from early allocation
- Add boot odometer claim
- Add privacy considerations for replay protection

E.14. From draft-ietf-rats-eat-12

- Make use of the JC<> generic to express CDDL for both JSON and CBOR
- Reorganize claims into 4 sections, particularly claims about the entity and about the token
- Nonce wording - say nonce is required and other improvements

- Clarify relationship of claims in evidence to results when forwarding
- Clarify manufacturer switching UEID types
- Add new section on the top-level token type that has CBOR-specific and JSON-specific CDDL since the top-level can't be handled with JC<>
- Remove definition of UCCS and UJCS, replacing it with a CDDL socket and mention of future token types
- Split the examples into payload and top level tokens since UCCS can't be used for examples any more (It was nice because you could see the payload claims in it easily, where you can't with CWT)
- DEB tag number is TBD rather than hard coded
- Add appendix with non-normative CDDL for a Claims-Set, CWT and JWT
- (Large reorganization of the document build and example verification makefile)
- Use CoAP content format ID to distinguish manifest and evidence formats instead of CBOR tag
- Added more examples, both CBOR and JSON
- All CDDL is validating against all examples
- Unassigned IANA requests are clearly TBD in the document (and have real values as is necessary in the example validation process)
- Improve security-level claim
- swresults claim is now measurement results claim
- substantial redesign of measurement results claim

Authors' Addresses

Laurence Lundblade
Security Theory LLC

EMail: lgl@securitytheory.com

Giridhar Mandyam
Qualcomm Technologies Inc.
5775 Morehouse Drive
San Diego, California
USA

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