RATS Working Group
Internet-Draft
Intended status: Standards Track
Expires: July 12, 2020

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

January 09, 2020

The Entity Attestation Token (EAT)
draft-ietf-rats-eat-02

Abstract

An Entity Attestation Token (EAT) provides a signed (attested) set of claims that describe state and characteristics of an entity, typically a device like a phone or an IoT device. These claims are used by a relying party to determine how much it wishes to trust the entity.

An EAT is either a CWT or JWT with some attestation-oriented claims. To a large degree, all this document does is extend CWT and JWT.

Contributing

TBD

Status of This Memo

Table of Contents

1. Introduction 3
1.1. CDDL, CWT and JWT 4
1.2. Entity Overview 4
1.3. EAT Operating Models 5
1.4. What is Not Standardized 6
1.4.1. Transmission Protocol 6
1.4.2. Signing Scheme 7
2. Terminology 7
3. The Claims Information Model 8
3.1. Token ID Claim (cti and jti) 8
3.2. Timestamp claim (iat) 8
3.3. Nonce Claim (nonce) 8
3.3.1. CDDL 9
3.4. Universal Entity ID Claim (ueid) 9
3.4.1. CDDL 11
3.5. Origination Claim (origination) 11
3.5.1. CDDL 12
3.6. OEM Identification by IEEE (oemid) 12
3.6.1. CDDL 12
3.7. The Security Level Claim (security_level) 12
3.7.1. CDDL 13
3.8. Secure Boot and Debug Enable State Claims (boot_state) . 13
3.8.1. Secure Boot Enabled 14
3.8.2. Debug Disabled 14
3.8.3. Debug Disabled Since Boot 14
3.8.4. Debug Permanent Disable 14
3.8.5. Debug Full Permanent Disable 14
3.8.6. CDDL 14
3.9. The Location Claim (location) 15
3.9.1. CDDL 15
3.10. The Age Claim (age) 15

3.11. The Uptime Claim (uptime) 15
3.11.1. CDDL 15
3.12. Nested EATs, the EAT Claim (nested_eat) 16
3.12.1. CDDL 16
3.13. The Submods Claim (submods) 16
3.13.1. The submod_name Claim 16
3.13.2. CDDL 17
4. Data Model 17
4.1. Common CDDL Types 17
4.2. CDDL for CWT-defined Claims 17
4.3. JSON 18
4.3.1. JSON Labels 18
4.3.2. JSON Interoperability 19
4.4. CBOR 19
4.4.1. Labels 19
4.4.2. CBOR Interoperability 20
4.5. Collected CDDL 21
5. IANA Considerations 22
5.1. Reuse of CBOR Web Token (CWT) Claims Registry 22
5.1.1. Claims Registered by This Document 22
6. Privacy Considerations 23
6.1. UEID Privacy Considerations 23
7. Security Considerations 24
8. References 24
8.1. Normative References 24
8.2. Informative References 25
Appendix A. Examples 27
A.1. Very Simple EAT 27
A.2. Example with Submodules, Nesting and Security Levels . . 27
Appendix B. Changes from Previous Drafts 28
B.1. From draft-mandyam-rats-eat-00 28
B.2. From draft-ietf-rats-eat-01 28
Authors' Addresses 28

1. Introduction

Remote device attestation is a fundamental service that allows a remote device such as a mobile phone, an Internet-of-Things (IoT) device, or other endpoint to prove itself to a relying party, a server or a service. This allows the relying party to know some characteristics about the device and decide whether it trusts the device.

Remote attestation is a fundamental service that can underlie other protocols and services that need to know about the trustworthiness of the device before proceeding. One good example is biometric authentication where the biometric matching is done on the device. The relying party needs to know that the device is one that is known

to do biometric matching correctly. Another example is content protection where the relying party wants to know the device will protect the data. This generalizes on to corporate enterprises that might want to know that a device is trustworthy before allowing corporate data to be accessed by it.

The notion of attestation here is large and may include, but is not limited to the following:

- Proof of the make and model of the device hardware (HW)
- Proof of the make and model of the device processor, particularly for security-oriented chips
- Measurement of the software (SW) running on the device
- Configuration and state of the device
- Environmental characteristics of the device such as its GPS location

1.1. CDDL, CWT and JWT

An EAT token is either a CWT as defined in [RFC8392] or a JWT as defined in [RFC7519]. This specification defines additional claims for entity attestation.

This specification uses CDDL, [RFC8610], as the primary formalism to define each claim. The implementor then interprets the CDDL to come to either the CBOR [RFC7049] or JSON [ECMAScript] representation. In the case of JSON, Appendix E of [RFC8610] is followed. Additional rules are given in Section 4.3.2 of this document where Appendix E is insufficient. (Note that this is not to define a general means to translate between CBOR and JSON, but only to define enough such that the claims defined in this document can be rendered unambiguously in JSON).

1.2. Entity Overview

An "entity" can be any device or device subassembly ("submodule") that can generate its own attestation in the form of an EAT. The attestation should be cryptographically verifiable by the EAT consumer. An EAT at the device-level can be composed of several submodule EAT’s. It is assumed that any entity that can create an EAT does so by means of a dedicated root-of-trust (RoT).

Modern devices such as a mobile phone have many different execution environments operating with different security levels. For example,

it is common for a mobile phone to have an "apps" environment that runs an operating system (OS) that hosts a plethora of downloadable apps. It may also have a TEE (Trusted Execution Environment) that is distinct, isolated, and hosts security-oriented functionality like biometric authentication. Additionally, it may have an eSE (embedded Secure Element) - a high security chip with defenses against HW attacks that can serve as a RoT. This device attestation format allows the attested data to be tagged at a security level from which it originates. In general, any discrete execution environment that has an identifiable security level can be considered an entity.

1.3. EAT Operating Models

At least the following three participants exist in all EAT operating models. Some operating models have additional participants.

The Entity. This is the phone, the IoT device, the sensor, the sub-assembly or such that the attestation provides information about.

The Manufacturer. The company that made the entity. This may be a chip vendor, a circuit board module vendor or a vendor of finished consumer products.

The Relying Party. The server, service or company that makes use of the information in the EAT about the entity.

In all operating models, the manufacturer provisions some secret attestation key material (AKM) into the entity during manufacturing. This might be during the manufacturer of a chip at a fabrication facility (fab) or during final assembly of a consumer product or any time in between. This attestation key material is used for signing EATs.

In all operating models, hardware and/or software on the entity create an EAT of the format described in this document. The EAT is always signed by the attestation key material provisioned by the manufacturer.

In all operating models, the relying party must end up knowing that the signature on the EAT is valid and consistent with data from claims in the EAT. This can happen in many different ways. Here are some examples.

- The EAT is transmitted to the relying party. The relying party gets corresponding key material (e.g. a root certificate) from the manufacturer. The relying party performs the verification.

- The EAT is transmitted to the relying party. The relying party transmits the EAT to a verification service offered by the manufacturer. The server returns the validated claims.

- The EAT is transmitted directly to a verification service, perhaps operated by the manufacturer or perhaps by another party. It verifies the EAT and makes the validated claims available to the relying party. It may even modify the claims in some way and re-sign the EAT (with a different signing key).

All these operating models are supported and there is no preference of one over the other. It is important to support this variety of operating models to generally facilitate deployment and to allow for some special scenarios. One special scenario has a validation service that is monetized, most likely by the manufacturer. In another, a privacy proxy service processes the EAT before it is transmitted to the relying party. In yet another, symmetric key material is used for signing. In this case the manufacturer should perform the verification, because any release of the key material would enable a participant other than the entity to create valid signed EATs.

1.4. What is Not Standardized

The following is not standardized for EAT, just the same they are not standardized for CWT or JWT.

1.4.1. Transmission Protocol

EATs may be transmitted by any protocol the same as CWTs and JWTs. For example, they might be added in extension fields of other protocols, bundled into an HTTP header, or just transmitted as files. This flexibility is intentional to allow broader adoption. This flexibility is possible because EAT’s are self-secured with signing (and possibly additionally with encryption and anti-replay). The transmission protocol is not required to fulfill any additional security requirements.

For certain devices, a direct connection may not exist between the EAT-producing device and the Relying Party. In such cases, the EAT should be protected against malicious access. The use of COSE and JOSE allows for signing and encryption of the EAT. Therefore, even if the EAT is conveyed through intermediaries between the device and Relying Party, such intermediaries cannot easily modify the EAT payload or alter the signature.

1.4.2. Signing Scheme

The term "signing scheme" is used to refer to the system that includes end-end process of establishing signing attestation key material in the entity, signing the EAT, and verifying it. This might involve key IDs and X.509 certificate chains or something similar but different. The term "signing algorithm" refers just to the algorithm ID in the COSE signing structure. No particular signing algorithm or signing scheme is required by this standard.

There are three main implementation issues driving this. First, secure non-volatile storage space in the entity for the attestation key material may be highly limited, perhaps to only a few hundred bits, on some small IoT chips. Second, the factory cost of provisioning key material in each chip or device may be high, with even millisecond delays adding to the cost of a chip. Third, privacy-preserving signing schemes like ECDAA (Elliptic Curve Direct Anonymous Attestation) are complex and not suitable for all use cases.

Over time to faciliate interoperability, some signing schemes may be defined in EAT profiles or other documents either in the IETF or outside.

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], COSE [RFC8152], and CWT [RFC8392].

Claim Name. The human-readable name used to identify a claim.

Claim Key. The CBOR map key or JSON name used to identify a claim.

Claim Value. The CBOR map or JSON object value representing the value of the claim.

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

Attestation Key Material (AKM). The key material used to sign the EAT token. If it is done symmetrically with HMAC, then this is a simple symmetric key. If it is done with ECC, such as an IEEE

DevID [IDevID], then this is the private part of the EC key pair. If ECDAA is used, (e.g., as used by Enhanced Privacy ID, i.e. EPID) then it is the key material needed for ECDAA.

3. The Claims Information Model

This section describes new claims defined for attestation. It also mentions several claims defined by CWT and JWT that are particularly important for EAT.

Note also: * Any claim defined for CWT or JWT may be used in an EAT including those in the CWT [IANA.CWT.Claims] and JWT IANA [IANA.JWT.Claims] claims registries.

- All claims are optional
- No claims are mandatory
- All claims that are not understood by implementations MUST be ignored

CDDL along with text descriptions is used to define the information model. Each claim is defined as a CDDL group (the group is a general aggregation and type definition feature of CDDL). In the data model, described in the Section 4, the CDDL groups turn into CBOR map entries and JSON name/value pairs.

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.

3.2. Timestamp claim (iat)

The "iat" claim defined in CWT and JWT is used to indicate the date-of-creation of the token.

3.3. Nonce Claim (nonce)

All EATs should have a nonce to prevent replay attacks. The nonce is generated by the relying party, the end consumer of the token. It is conveyed to the entity over whatever transport is in use before the token is generated and then included in the token as the nonce claim.

This documents the nonce claim for registration in the IANA CWT claims registry. This is equivalent to the JWT nonce claim that is already registered.

The nonce must be at least 8 bytes (64 bits) as fewer are unlikely to be secure. A maximum of 64 bytes is set to limit the memory a constrained implementation uses. This size range is not set for the already-registered JWT nonce, but it should follow this size recommendation when used in an EAT.

3.3.1. CDDL

```
nonce_claim = (
    nonce => bstr .size (8..64)
)
```

3.4. Universal Entity ID Claim (ueid)

UEID’s identify individual manufactured entities / devices such as a mobile phone, a water meter, a Bluetooth speaker or a networked security camera. It may identify the entire device or a submodule or subsystem. It does not identify types, models or classes of devices. It is akin to a serial number, though it does not have to be sequential.

UEID’s 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 device to keep devices distinct between manufacturers).

There are privacy considerations for UEID’s. See Section 6.1.

The UEID should be permanent. It should never change for a given device / entity. In addition, it should not be reprogrammable. UEID’s are variable length. The recommended maximum is 33 bytes (1 type byte and 256 bits). The recommended minimum is 17 bytes (1 type and 128 bits) because fewer bytes endanger the universal uniqueness.

When the entity constructs the UEID, the first byte is a type and the following bytes the ID for that type. Several types are allowed to accommodate different industries and different manufacturing processes and to give options to avoid paying fees for certain types of manufacturer registrations.

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

Type Byte	Type Name	Specification
0x01	RAND	This is a 128- to 256-bit random number generated once and stored in the device. This may be constructed by concatenating enough identifiers to be universally unique and then feeding the concatenation through a cryptographic hash function. It may also be a cryptographic quality random number generate once at the beginning of the life of the device and stored.
0x02	IEEE EUI	This makes use of the IEEE company identification registry. An EUI is made up of an OUI and OUI-36 or a CID, different registered company identifiers, and some unique per-device identifier. EUIs are often the same as or similar to MAC addresses. (Note that while devices with multiple network interfaces may have multiple MAC addresses, there is only one UEID for a device) TODO: normative references to IEEE.
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 a binary integer over 48 bits. The IMEI value encoded SHALL NOT include Luhn checksum or SVN information.
0x04	EUI-48	This is a 48-bit identifier formed by concatenating the 24-bit OUI with a 24-bit identifier assigned by the organisation that purchased the OUI.
0x05	EUI-60	This is a 60-bit identifier formed by concatenating the 24-bit OUI with a 36-bit identifier assigned by the organisation that purchased the OUI.
0x06	EUI-64	This is a 64-bit identifier formed by concatenating the 24-bit OUI with a 40-bit identifier assigned by the organisation that purchased the OUI.

Table 1: UEID Composition Types

UEID’s 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 (the relying party) 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 device. Instead they should use the OUI claim that is defined elsewhere. The reasons for this are:

- UEIDs types may vary freely from one manufacturer to the next.
- New types of UEIDs may be created. For example, a type 0x07 UEID may be created based on some other manufacturer registration scheme.
- Device manufacturers are allowed to change from one type of UEID to another anytime they want. For example, they may find they can optimize their manufacturing by switching from type 0x01 to type 0x02 or vice versa. The main requirement on the manufacturer is that UEIDs be universally unique.

3.4.1. CDDL

```
ueid_claim = (
ueid: bstr )
```

3.5. Origination Claim (origination)

This claim describes the parts of the device or entity that are creating the EAT. Often it will be tied back to the device or chip manufacturer. The following table gives some examples:

Name	Description
Acme-TEE	The EATs are generated in the TEE authored and configured by "Acme"
Acme-TPM	The EATs are generated in a TPM manufactured by "Acme"
Acme-Linux-Kernel	The EATs are generated in a Linux kernel configured and shipped by "Acme"
Acme-TA	The EATs are generated in a Trusted Application (TA) authored by "Acme"

TODO: consider a more structure approach where the name and the URI and other are in separate fields.

TODO: This needs refinement. It is somewhat parallel to issuer claim in CWT in that it describes the authority that created the token.

3.5.1. CDDL

```
origination_claim = (
origination: string_or_uri )
```

3.6. OEM Identification by IEEE (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 services [OUI.Lookup]

Companies that have more than one of these IDs or MAC address blocks should pick one and prefer that for all their devices.

Commonly, these are expressed in Hexadecimal Representation [IEEE.802-2001] also called the Canonical format. When this claim is encoded 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. For JSON encoded tokens, this is further base64url encoded.

3.6.1. CDDL

```
oemid_claim = (
oemid: bstr )
```

3.7. The Security Level Claim (security_level)

EATs have a claim that roughly characterizes the device / entities ability to defend against attacks aimed at capturing the signing key, forging claims and at forging EATs. This is done by roughly defining four security levels as described below. This is similar to the security levels defined in the Metadata Service defined by the Fast Identity Online (FIDO) Alliance (TODO: reference).

These claims describe security environment and countermeasures available on the end-entity / client device where the attestation key reside and the claims originate.

1 - Unrestricted There is some expectation that implementor will protect the attestation signing keys at this level. Otherwise the EAT provides no meaningful security assurances.

2- Restricted Entities at this level should not be general-purpose operating environments that host features such as app download systems, web browsers and complex productivity applications. It is akin to the Secure Restricted level (see below) without the security orientation. Examples include a Wi-Fi subsystem, an IoT camera, or sensor device.

3 - Secure Restricted Entities at this level must meet the criteria defined by FIDO Allowed Restricted Operating Environments (TODO: reference). Examples include TEE's and schemes using virtualization-based security. Like the FIDO security goal, security at this level is aimed at defending well against large-scale network / remote attacks against the device.

4 - Hardware Entities at this level must include substantial defense against physical or electrical attacks against the device itself. It is assumed any potential attacker has captured the device and can disassemble it. Example include TPMs and Secure Elements.

This claim is not intended as a replacement for a proper end-device security certification schemes such as those based on FIPS (TODO: reference) or those based on Common Criteria (TODO: reference). The claim made here is solely a self-claim made by the Entity Originator.

3.7.1. CDDL

```
security_level_type = (
unrestricted: 1,
restricted: 2,
secure_restricted: 3,
hardware: 4
)

security_level_claim = (
security_level: security_level_type )
```

3.8. Secure Boot and Debug Enable State Claims (boot_state)

This claim is an array of five Boolean values indicating the boot and debug state of the entity.

3.8.1. Secure Boot Enabled

This indicates whether secure boot is enabled either for an entire device or an individual submodule. If it appears at the device level, then this means that secure boot is enabled for all submodules. Secure boot enablement allows a secure boot loader to authenticate software running either in a device or a submodule prior allowing execution.

3.8.2. Debug Disabled

This indicates whether debug capabilities are disabled for an entity (i.e. value of 'true'). Debug disablement is considered a prerequisite before an entity is considered operational.

3.8.3. Debug Disabled Since Boot

This claim indicates whether debug capabilities for the entity were not disabled in any way since boot (i.e. value of 'true').

3.8.4. Debug Permanent Disable

This claim indicates whether debug capabilities for the entity are permanently disabled (i.e. value of 'true'). This value can be set to 'true' also if only the manufacturer is allowed to enabled debug, but the end user is not.

3.8.5. Debug Full Permanent Disable

This claim indicates whether debug capabilities for the entity are permanently disabled (i.e. value of 'true'). This value can only be set to 'true' if no party can enable debug capabilities for the entity. Often this is implemented by blowing a fuse on a chip as fuses cannot be restored once blown.

3.8.6. CDDL

```
boot_state_type = [
    secure_boot_enabled=> bool,
    debug_disabled=> bool,
    debug_disabled_since_boot=> bool,
    debug_permanent_disable=> bool,
    debug_full_permanent_disable=> bool
]

boot_state_claim = (
boot_state: boot_state_type
)
```

3.9. The Location Claim (location)

The location claim is a CBOR-formatted object that describes the location of the device entity from which the attestation originates. It is comprised of a map of additional sub claims that represent the actual location coordinates (latitude, longitude and altitude). The location coordinate claims are consistent with the WGS84 coordinate system [WGS84]. In addition, a sub claim providing the estimated accuracy of the location measurement is defined.

3.9.1. CDDL

```
location_type = {
    latitude => number,
    longitude => number,
    altitude => number,
    accuracy => number,
    altitude_accuracy => number,
    heading => number,
    speed => number
}

location_claim = (
location: location_type )
```

3.10. The Age Claim (age)

The "age" claim contains a value that represents the number of seconds that have elapsed since the token was created, measurement was made, or location was obtained. Typical attestable values are sent as soon as they are obtained. However, in the case that such a value is buffered and sent at a later time and a sufficiently accurate time reference is unavailable for creation of a timestamp, then the age claim is provided.

```
age_claim = (
age: uint)
```

3.11. The Uptime Claim (uptime)

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

3.11.1. CDDL

```
uptime_claim = (
uptime: uint )
```

3.12. Nested EATs, the EAT Claim (nested_eat)

It is allowed for one EAT to be embedded in another. This is for complex devices that have more than one subsystem capable of generating an EAT. For example, one might be the device-wide EAT that is low to medium security and another from a Secure Element or similar that is high security.

The contents of the "nested_eat" claim must be a fully signed, optionally encrypted, EAT token.

3.12.1. CDDL

```
nested_eat_claim = (
nested_eat: nested_eat_type)
```

A nested_eat_type is defined in words rather than CDDL. It is either a full CWT or JWT including the COSE or JOSE signing.

3.13. The Submods Claim (submods)

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

The claims for each these can be grouped together in a submodule.

Specifically, the "submods" claim is an array. Each item in the array is a CBOR map containing all the claims for a particular submodule.

The security level of the submod is assumed to be at the same level as the main entity unless there is a security level claim in that submodule indicating otherwise. The security level of a submodule can never be higher (more secure) than the security level of the EAT it is a part of.

3.13.1. The submod_name Claim

Each submodule should have a submod_name claim that is descriptive name. This name should be the CBOR txt type.

3.13.2. CDDL

In the following a generic_claim_type is any CBOR map entry or JSON name/value pair.

```
submod_name_type = (
submod_name: tstr )

submods_type = [ * submod_claims ]

submod_claims = {
    submod_name_type,
    * generic_claim_type
}

submods_claim = (
submods: submod_type )
```

4. Data Model

This makes use of the types defined in CDDL Appendix D, Standard Prelude.

4.1. Common CDDL Types

```
string_or_uri = #6.32(tstr) / tstr; See JSON section below for JSON encoding
 of string_or_uri
```

4.2. CDDL for CWT-defined Claims

This section provides CDDL for the claims defined in CWT. It is non-normative as [RFC8392] is the authoritative definition of these claims.

```
   cwt_claim = (
       issuer_claim //
       subject_claim //
       audience_claim //
       expiration_claim //
       not_before_claim //
       issued_at_calim //
       cwt_id_claim
   )

   issuer_claim = (
   issuer: string_or_uri )

   subject_claim = (
   subject: string_or_uri )

   audience_claim = (
   audience: string_or_uri )

   expiration_claim = (
   expiration: time )

   not_before_claim = (
   not_before: time )

   issued_at_calim = (
   issued_at: time )

   cwt_id_claim = (
   cwt_id: bstr )

   issuer = 1
   subject = 2
   audience = 3
   expiration = 4
   not_before = 5
   issued_at = 6
   cwt_id = 7
```

4.3. JSON

4.3.1. JSON Labels

```
ueid = "ueid"
origination = "origination"
oemid = "oemid"
security_level = "security_level"
boot_state = "boot_state"
location = "location"
age = "age"
uptime = "uptime"
nested_eat = "nested_eat"
submods = "submods"

latitude = "lat""
longitude = "long""
altitude = "alt"
accuracy = "accry"
altitude_accuracy = "alt_accry"
heading = "heading"
speed = "speed"
```

4.3.2. JSON Interoperability

JSON should be encoded per RFC 8610 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].

4.4. CBOR

4.4.1. Labels

```
ueid = 8
origination = 9
oemid = 10
security_level = 11
boot_state = 12
location = 13
age = 14
uptime = 15
nested_eat = 16
submods = 17
submod_name = 18
nonce = 19

latitude = 1
longitude = 2
altitude = 3
accuracy = 4
altitude_accuracy = 5
heading = 6
speed = 7
```

4.4.2. CBOR Interoperability

Variations in the CBOR serializations supported in CBOR encoding and decoding are allowed and suggests that CBOR-based protocols specify how this variation is handled. This section specifies what formats MUST be supported in order to achieve interoperability.

The assumption is that the entity is likely to be a constrained device and relying party is likely to be a very capable server. The approach taken is that the entity generating the token can use whatever encoding it wants, specifically encodings that are easier to implement such as indefinite lengths. The relying party receiving the token must support decoding all encodings.

These rules cover all types used in the claims in this document. They also are recommendations for additional claims.

Canonical CBOR encoding, Preferred Serialization and Deterministically Encoded CBOR are explicitly NOT required as they would place an unnecessary burden on the entity implementation, particularly if the entity implementation is implemented in hardware.

- Integer Encoding (major type 0, 1) - The entity may use any integer encoding allowed by CBOR. The server MUST accept all integer encodings allowed by CBOR.

- String Encoding (major type 2 and 3) - The entity can use any string encoding allowed by CBOR including indefinite lengths. It may also encode the lengths of strings in any way allowed by CBOR. The server must accept all string encodings.

- Major type 2, bstr, SHOULD be have tag 21 to indicate conversion to base64url in case that conversion is performed.

- Map and Array Encoding (major type 4 and 5) - The entity can use any array or map encoding allowed by CBOR including indefinite lengths. Sorting of map keys is not required. Duplicate map keys are not allowed. The server must accept all array and map encodings. The server may reject maps with duplicate map keys.

- Date and Time - The entity should send dates as tag 1 encoded as 64-bit or 32-bit integers. The entity may not send floating-point dates. The server must support tag 1 epoch-based dates encoded as 64-bit or 32-bit integers. The entity may send tag 0 dates, however tag 1 is preferred. The server must support tag 0 UTC dates.

- URIs - URIs should be encoded as text strings and marked with tag 32.

- Floating Point - The entity may use any floating-point encoding. The relying party must support decoding of all types of floating-point.

- Other types - Use of Other types like bignums, regular expressions and such, SHOULD NOT be used. The server MAY support them but is not required to so interoperability is not guaranteed.

4.5. Collected CDDL

A generic_claim is any CBOR map entry or JSON name/value pair.

```
eat_claims = {  ; the top-level payload that is signed using COSE or JOSE
        * claim
}

claim = (
    ueid_claim //
    origination_claim //
    oemid_claim //
    security_level_claim //
    boot_state_claim //
    location_claim //
    age_claim //
    uptime_claim //
    nested_eat_claim //
    cwt_claim //
    generic_claim_type //
    )
```

TODO: copy the rest of the CDDL here (wait until the CDDL is more settled so as to avoid copying multiple times)

5. IANA Considerations

5.1. Reuse of CBOR Web Token (CWT) Claims Registry

Claims defined for EAT are compatible with those of CWT so the CWT Claims Registry is re used. No new IANA registry is created. All EAT claims should be registered in the CWT and JWT Claims Registries.

5.1.1. Claims Registered by This Document

- Claim Name: UEID
- Claim Description: The Universal Entity ID
- JWT Claim Name: N/A
- Claim Key: 8
- Claim Value Type(s): byte string
- Change Controller: IESG
- Specification Document(s): *this document*

TODO: add the rest of the claims in here

6. 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.

6.1. UEID Privacy Considerations

A UEID is usually not privacy-preserving. Any set of relying parties that receives tokens that happen to be from a single device will be able to know the tokens are all from the same device and be able to track the device. Thus, in many usage situations ueid violates governmental privacy regulation. In other usage situations 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.

There are several strategies that can be used to still be able to put UEID's in tokens:

- The device obtains explicit permission from the user of the device to use the UEID. 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.

- The UEID is used only in a particular context or particular use case. It is used only by one relying party.

- The device authenticates the relying party and generates a derived UEID just for that particular relying party. For example, the relying party could prove their identity cryptographically to the device, then the device generates a UEID just for that relying party by hashing a proofed relying party ID with the main device UEID.

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

7. Security Considerations

TODO: Perhaps this can be the same as CWT / COSE, but not sure yet because it involves so much entity / device security that those do not.

8. References

8.1. Normative References

[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>.

[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>.

[RFC7049] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", RFC 7049, DOI 10.17487/RFC7049, October 2013, <https://www.rfc-editor.org/info/rfc7049>.

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

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

[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>.

[TIME_T] The Open Group Base Specifications, "Vol. 1: Base Definitions, Issue 7", Section 4.15 'Seconds Since the Epoch', IEEE Std 1003.1, 2013 Edition, 2013, <http://pubs.opengroup.org/onlinepubs/9699919799/basedefs/V1_chap04.html#tag_04_15>.

[WGS84] National Imagery and Mapping Agency, "National Imagery and Mapping Agency Technical Report 8350.2, Third Edition", 2000, <http://earth-info.nga.mil/GandG/publications/tr8350.2/wgs84fin.pdf>.

8.2. Informative References

[ASN.1] International Telecommunication Union, "Information Technology -- ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)", ITU-T Recommendation X.690, 1994.

[ECMAScript] "Ecma International, "ECMAScript Language Specification, 5.1 Edition", ECMA Standard 262", June 2011, <http://www.ecma-international.org/ecma-262/5.1/ECMA-262.pdf>.

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

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

[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>.

[Webauthn]
Worldwide Web Consortium, "Web Authentication: A Web API for accessing scoped credentials", 2016.

Appendix A. Examples

A.1. Very Simple EAT

This is shown in CBOR diagnostic form. Only the payload signed by COSE is shown.

```
{
   / nonce (cti) /            7:h'948f8860d13a463e8e',
   / UEID /                   8:h'0198f50a4ff6c05861c8860d13a638ea4fe2f',
   / boot_state /            12:{true, true, true, true, false}
   / time stamp (iat) /       6:1526542894,
}
```

A.2. Example with Submodules, Nesting and Security Levels

```
{
   / nonce /                  7:h'948f8860d13a463e8e',
   / UEID /                   8:h'0198f50a4ff6c05861c8860d13a638ea4fe2f',
   / boot_state /            12:{true, true, true, true, false}
   / time stamp (iat) /       6:1526542894,
   / seclevel /              11:3, / secure restricted OS /

   / submods / 17:
      [
         / 1st submod, an Android Application / {
           / submod_name /   18:'Android App "Foo"',
           / seclevel /      11:1, / unrestricted /
           / app data /  -70000:'text string'
         },
         / 2nd submod, A nested EAT from a secure element / {
           / submod_name / 18:'Secure Element EAT',
           / eat /         16:61( 18(
              / an embedded EAT / [ /...COSE_Sign1 bytes with payload.../ ]
                         ))
         }
         / 3rd submod, information about Linux Android / {
            / submod_name/ 18:'Linux Android',
            / seclevel /   11:1, / unrestricted /
            / custom - release / -80000:'8.0.0',
            / custom - version / -80001:'4.9.51+'
         }
      ]
}
```

Appendix B. 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.

B.1. From draft-mandyam-rats-eat-00

This is a fairly large change in the orientation of the document, but not 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

B.2. From draft-ietf-rats-eat-01

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

Authors' Addresses

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

Phone: +1 858 651 7200
EMail: mandyam@qti.qualcomm.com

Laurence Lundblade
Security Theory LLC

EMail: lgl@island-resort.com

Miguel Ballesteros
Qualcomm Technologies Inc.
5775 Morehouse Drive
San Diego, California
USA

Phone: +1 858 651 4299
EMail: mballest@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