RATS Working Group   G. Mandyam
Internet-Draft   Qualcomm Technologies Inc.
Intended status: Standards Track   L. Lundblade
Expires: December 24, 2019   Security Theory LLC
                                                      M. Ballesteros
                                                        J. O'Donoghue
                                             Qualcomm Technologies Inc.
                                                        June 22, 2019

The Entity Attestation Token (EAT)
draft-ietf-rats-eat-00

Abstract

   An attestation format based on concise binary object representation
   (CBOR) is proposed that is suitable for inclusion in a CBOR Web Token
   (CWT), know as the Entity Attestation Token (EAT). The associated
   data can be used by a relying party to assess the security state of a
   remote device or module.

Contributing

   TBD

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 December 24, 2019.

Copyright Notice

Table of Contents

1. Introduction

   Remote device attestation is 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:

   o Proof of the make and model of the device hardware (HW)

o Proof of the make and model of the device processor, particularly
   for security oriented chips

o Measurement of the software (SW) running on the device

o Configuration and state of the device

o Environmental characteristics of the device such as its GPS
   location

The required data format should be general purpose and extensible so
that it can work across many use cases. This is why CBOR (see
[RFC7049]) was chosen as the format -- it already supports a rich set
of data types, and is both expressive and extensible. It translates
well to JSON for good interoperation with web technology. It is
compact and can work on very small IoT device. The format proposed
here is small enough that a limited version can be implemented in
pure hardware gates with no software at all. Moreover, the
attestation data is defined in the form of claims that is the same as
CBOR Web Token (CWT, see [RFC8392]). This is the motivation for
defining the Entity Attestation Token, i.e. EAT.

1.1. 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.

1.2. Use of CBOR and COSE

Fundamentally this attestation format is a verifiable data format.
It is a collection of data items that can be signed by an attestation
key, hashed, and/or encrypted. As per Section 7 of [RFC8392], the
verification method is in the CWT using the CBOR Object Signing and
Encryption (COSE) methodology (see [RFC8152]).

In addition, the reported attestation data could be determined within
the secure operating environment or written to it from an external
and presumably less trusted entity on the device. In either case,
the source of the reported data must be identifiable by the relying
party.

This attestation format is a single relatively simple signed message.
It is designed to be incorporated into many other protocols and many
other transports. It is also designed such that other SW and apps
can add their own data to the message such that it is also attested.

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.

o 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.

o 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.

o 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).

This standard supports all these operating models and does not prefer
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

1.4.1. Transmission Protocol

EATs may be transmitted by any protocol. 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 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.

Eventually some form of standardization of the signing scheme may be
required. This might come in the form of another standard that adds
to this document, or when there is clear convergence on a small
number of signing schemes this standard can be updated.

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].

StringOrURI. The "StringOrURI" term in this specification has the
   same meaning and processing rules as the JWT "StringOrURI" term
   defined in Section 2 of [RFC7519], except that it is represented
   as a CBOR text string instead of a JSON text string.

NumericDate. The "NumericDate" term in this specification has the
   same meaning and processing rules as the JWT "NumericDate" term
   defined in Section 2 of [RFC7519], except that it is represented
   as a CBOR numeric date (from Section 2.4.1 of [RFC7049]) instead

of a JSON number. The encoding is modified so that the leading
tag 1 (epoch-based date/time) MUST be omitted.

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

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

Claim Value. The CBOR map value representing the value of the claim.

CWT Claims Set. The CBOR map that contains the claims conveyed by
the CWT.

FloatOrNumber. The "FloatOrNumber" term in this specification is the
type of a claim that is either a CBOR positive integer, negative
integer or floating point number.

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

3.1. Universal Entity ID (UEID) Claim

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.

It is identified by Claim Key X (X is TBD).

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).

The UEID should be permanent. It should never change for a given
device / entity. In addition, it should not be reprogrammable.

UEID's are binary byte-strings (resulting in a smaller size than text
strings). When handled in text-based protocols, they should be
base-64 encoded.

UEID's are variable length with a maximum size of 33 bytes (1 type
byte and 256 bits). A receivers of a token with UEIDs may reject the
token if a UEID is larger than 33 bytes.

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.

A UEID is a byte string. From the consumer's view (the rely party)
it is opaque with no bytes having any special meaning.

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.

```
+------+--------+---------------------------------------------------+
| Type | Type   | Specification                                     |
| Byte | Name   |                                                   |
+------+--------+---------------------------------------------------+
| 0x01 | GUID   | This is a 128 to 256 bit random number generated  |
|      |        | once and stored in the device. The GUID may be    |
|      |        | constructed from various identifiers on the       |
|      |        | device using a hash function or it may be just    |
|      |        | the raw random number.                            |
| 0x02 | IEEE   | This makes use of the IEEE company identification |
|      | EUI    | 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 six 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

   The consumer (the Relying Party) of a UEID should 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:

   o UEIDs types may vary freely from one manufacturer to the next.

o New types of UEIDs may be created. For example a type 0x04 UEID
   may be created based on some other manufacturer registration
   scheme.

o 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.

3.2. Origination (origination) Claims

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"            |
+-------------------+------------------------------------------------+
```

The claim is represented by Claim Key X+1. It is type StringOrURI.

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.

3.3. OEM identification by IEEE OUI

This claim identifies a device OEM by the IEEE OUI. Reference TBD.
It is a byte string representing the OUI in binary form in network
byte order (TODO: confirm details).

Companies that have more than one IEEE OUI registered with IEEE
should pick one and prefer that for all their devices.

Note that the OUI is in common use as a part of MAC Address. This
claim is only the first bits of the MAC address that identify the

manufacturer. The IEEE maintains a registry for these in which many
companies participate. This claim is represented by Claim Key TBD.

3.4. Security Level (seclevel) Claim

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 definied 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.

This claim is identified by Claim Key X+2. The value is an integer
between 1 and 4 as defined below.

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 WiFi subsystem, an IoT
   camera, or sensor device.

3 - Secure Restricted Entities at this level must meet the critera
   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.

[3.5](#). Nonce (nonce) Claim

The "nonce" (Nonce) claim represents a random value that can be used to avoid replay attacks. This would be ideally generated by the CWT consumer. This value is intended to be a CWT companion claim to the existing JWT claim **_IANAJWT_ (TODO: fix this reference). The nonce claim is identified by Claim Key X+3.

[3.6](#). Secure Boot and Debug Enable State Claims

[3.6.1](#). Secure Boot Enabled (secbootenabled) Claim

The "secbootenabled" (Secure Boot Enabled) claim represents a boolean value that 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. This claim is identified by Claim Key X+4.

[3.6.2](#). Debug Disabled (debugdisabled) Claim

The "debugdisabled" (Debug Disabled) claim represents a boolean value that 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. This claim is identified by Claim Key X+5.

[3.6.3](#). Debug Disabled Since Boot (debugdisabledsincebboot) Claim

The "debugdisabledsinceboot" (Debug Disabled Since Boot) claim represents a boolean value that indicates whether debug capabilities for the entity were not disabled in any way since boot (i.e. value of 'true'). This claim is identified by Claim Key X+6.

[3.6.4](#). Debug Permanent Disable (debugpermanentdisable) Claim

The "debugpermanentdisable" (Debug Permanent Disable) claim represents a boolean value that 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. This claim is identified by Claim Key X+7.

3.6.5. Debug Full Permanent Disable (debugfullpermanentdisable) Claim

   The "debugfullpermanentdisable" (Debug Full Permanent Disable) claim
   represents a boolean value that 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. This claim is
   identified by Claim Key X+8.

3.7. Location (loc) Claim

   The "loc" (location) claim is a CBOR-formatted object that describes
   the location of the device entity from which the attestation
   originates. It is identified by Claim Key X+10. It is comprised of
   an array of additional subclaims 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 subclaim providing the estimated accuracy of
   the location measurement is defined.

3.7.1. lat (latitude) claim

   The "lat" (latitude) claim contains the value of the device location
   corresponding to its latitude coordinate. It is of data type
   FloatOrNumber and identified by Claim Key X+11.

3.7.2. long (longitude) claim

   The "long" (longitude) claim contains the value of the device
   location corresponding to its longitude coordinate. It is of data
   type FloatOrNumber and identified by Claim Key X+12.

3.7.3. alt (altitude) claim

   The "alt" (altitude) claim contains the value of the device location
   corresponding to its altitude coordinate (if available). It is of
   data type FloatOrNumber and identified by Claim Key X+13.

3.7.4. acc (accuracy) claim

   The "acc" (accuracy) claim contains a value that describes the
   location accuracy. It is non-negative and expressed in meters. It
   is of data type FloatOrNumber and identified by Claim Key X+14.

3.7.5. altacc (altitude accuracy) claim

The "altacc" (altitude accuracy) claim contains a value that
describes the altitude accuracy. It is non-negative and expressed in
meters. It is of data type FloatOrNumber and identified by Claim Key
X+15.

3.7.6. heading claim

The "heading" claim contains a value that describes direction of
motion for the entity. Its value is specified in degrees, between 0
and 360. It is of data type FloatOrNumber and identified by Claim
Key X+16.

3.7.7. speed claim

The "speed" claim contains a value that describes the velocity of the
entity in the horizontal direction. Its value is specified in
meters/second and must be non-negative. It is of data type
FloatOrNumber and identified by Claim Key X+17.

3.8. ts (timestamp) claim

The "ts" (timestamp) claim contains a timestamp derived using the
same time reference as is used to generate an "iat" claim (see
Section 3.1.6 of [RFC8392]). It is of the same type as "iat"
(integer or floating-point), and is identified by Claim Key X+18. It
is meant to designate the time at which a measurement was taken, when
a location was obtained, or when a token was actually transmitted.
The timestamp would be included as a subclaim under the "submod" or
"loc" claims (in addition to the existing respective subclaims), or
at the device level.

3.9. age claim

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. It is identified by Claim Key X+19.

3.10. uptime claim

The "uptime" claim contains a value that represents the number of
seconds that have elapsed since the entity or submod was last booted.
It is identified by Claim Key X+20.

3.11. The submods Claim

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., WiFi and cellular). It may have
sub systems 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. It is identified by Claim Key X+22.

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.

3.11.1. The submod_name Claim

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

3.11.2. Nested EATs, the eat Claim

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. Typically one will 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 "eat" claim must be a fully signed, optionally
encrypted, EAT token. It is identified by Claim Key X+23.

4. CBOR Interoperability

EAT is a one-way protocol. It only defines a single message that
goes from the entity to the server. The entity implementation will
often be in a contained environment with little RAM and the server
will usually not be. The following requirements for interoperability
take that into account. The entity can generally use whatever
encoding it wants. The server is required to support just about
every encoding.

Canonical CBOR encoding is explicitly NOT required as it would place
an unnecessary burden on the entity implementation.

4.1. Integer Encoding (major type 0 and 1)

The entity may use any integer encoding allowed by CBOR. The server
MUST accept all integer encodings allowed by CBOR.

4.2. 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, 22 or 23 to indicate
conversion to base64 or such when converting to JSON.

4.3. 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.

4.4. 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.

4.5. URIs

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

4.6. Floating Point

Encoding data in floating point is to be used only if necessary.
Location coordinates are always in floating point. The server must
support decoding of all types of floating point.

4.7. Other types

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

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. New new IANA registry is created. All
   EAT claims should be registered in the CWT Claims Registry.

5.1.1. Claims Registered by This Document

   o Claim Name: UEID

   o Claim Description: The Universal Entity ID

   o JWT Claim Name: N/A

   o Claim Key: X

   o Claim Value Type(s): byte string

   o Change Controller: IESG

   o Specification Document(s): *this document*

   TODO: add the rest of the claims in here

5.2. EAT CBOR Tag Registration

   How an EAT consumer determines whether received CBOR-formatted data
   actually represents a valid EAT is application-dependent, much like a
   CWT. For instance, a specific MIME type associated with the EAT such
   as "application/eat" could be sufficient for identification of the
   EAT. Note however that EAT's can include other EAT's (e.g. a device
   EAT comprised of several submodule EAT's). In this case, a CBOR tag
   dedicated to the EAT will be required at least for the submodule
   EAT's and the tag must be a valid CBOR tag. In other words - the EAT
   CBOR tag can optionally prefix a device-level EAT, but a EAT CBOR tag
   must always prefix a submodule EAT. The proposed EAT CBOR tag is 71.

5.2.1. Tag Registered by This Document

   o CBOR Tag: 71

   o Data Item: Entity Attestation Token (EAT)

   o Semantics: Entity Attestation Token (CWT), as defined in
      this_doc

   o Reference: *this_doc*

   o Point of Contact: Giridhar Mandyam, mandyam@qti.qualcomm.com

[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:

   o 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.

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

   o 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

   [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>.

   [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>.

   [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.

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

[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 /                 11:h'948f8860d13a463e8e',
    / UEID /                   8:h'0198f50a4ff6c05861c8860d13a638ea4fe2f',
    / secbootenabled /        13:true,
    / debugpermanentdisable / 15:true,
    / ts /                    21:1526542894,
}
```

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

```
{
    / nonce /                 11:h'948f8860d13a463e8e',
    / UEID /                   8:h'0198f50a4ff6c05861c8860d13a638ea4fe2f',
    / secbootenabled /        13:true,
    / debugpermanentdisable / 15:true,
    / ts /                    21:1526542894,
    / seclevel /              10:3, / secure restriced OS /

    / submods / 30:
      [
         / 1st submod, an Android Application / {
           / submod_name /   30:'Android App "Foo"',
           / seclevel /      10:1, / unrestricted /
           / app data /  -70000:'text string'
         },
         / 2nd submod, A nested EAT from a secure element / {
           / submod_name / 30:'Secure Element EAT',
           / eat /          31:71( 18(
              / an embedded EAT / [ /...COSE_Sign1 bytes with payload.../ ]
                        ))
         }
         / 3rd submod, information about Linux Android / {
            / submod_name/ 30:'Linux Android',
            / seclevel /   10:1, / unrestricted /
            / custom - release / -80000:'8.0.0',
            / custom - version / -80001:'4.9.51+'
         }
      ]
}
```

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