Workgroup: RATS Working Group
Internet-Draft: draft-ietf-rats-eat-latest
Published: 21 January 2021
Intended Status: Standards Track
Expires: 25 July 2021
Authors: G. Mandyam
Qualcomm Technologies Inc.
L. Lundblade
Security Theory LLC
M. Ballesteros
Qualcomm Technologies Inc.
J. O'Donoghue
Qualcomm Technologies Inc.

The Entity Attestation Token (EAT)

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

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 25 July 2021.

Copyright Notice

Copyright (c) 2021 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (https://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

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

TODO: mention use for Attestation Evidence and Results.

1.1. CWT, JWT and UCCS

For flexibility and ease of imlpementation in a wide variety of environments, EATs can be either CBOR [RFC8949] or JSON [ECMAScript] format. This specification simultaneously describes both formats.

An EAT is either a CWT as defined in [RFC8392], a UCCS as defined in [UCCS.Draft], or a JWT as defined in [RFC7519]. This specification extends those specifications with additional claims for attestation.

The identification of a protocol element as an EAT, whether CBOR or JSON format, follows the general conventions used by CWT, JWT and UCCS. 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.

1.2. CDDL

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 [RFC8949] or JSON [ECMAScript] representation. In the case of JSON, Appendix E of [RFC8610] is followed. Additional rules are given in Section 6.3.2 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).

The CWT specification was authored before CDDL was available and did not use it. This specification includes a CDDL definition of most of what is described in [RFC8392].

1.3. 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.

1.4. EAT Operating Models

TODO: Rewrite (or eliminate) this section in light of the RATS architecture draft.

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.

1.5. What is Not Standardized

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

1.5.1. 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.

1.5.2. 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 value portion of the claim. A claim value can be any CBOR data item or JSON value.

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

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

There are no default values or meanings assigned to absent claims other than they are not reported. The reason for a claim's absence may be the implementation not supporting the claim, an inability to determine its value, or a preference to report in a different way such as a proprietary claim.

CDDL along with text descriptions is used to define each claim indepdent of encoding. Each claim is defined as a CDDL group (the group is a general aggregation and type definition feature of CDDL). In the encoding section Section 6, the CDDL groups turn into CBOR map entries and JSON name/value pairs.

TODO: add paragraph here about use for Attestation Evidence and for Results.

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, 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. No token may contain an iat claim in float-point format. Any recipient of a token with a floating-point format iat claim may 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.

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.

Multiple nonces are allowed to accommodate multistage verification and consumption.

3.3.1. nonce CDDL

```
nonce-type = bstr .size (8..64)

nonce-claim = (
    nonce => nonce-type / [ 2* nonce-type ]
)
```

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 8.1.

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. All implementations MUST be able to receive UEID's that are 33 bytes long (1 type byte and 256 bits). The recommended maximum sent is also 33 bytes.

When the entity constructs the UEID, the first byte is a type and the following bytes are 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, 192 or 256 bit random number generated once and stored in the device. 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 device and stored. It may not be smaller than 128 bits.
0x02	IEEE EUI	This makes use of 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-device 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 devices with multiple network interfaces may have multiple MAC addresses, there is only one UEID for a device) [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. [ThreeGPP.IMEI]

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 oemid 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. ueid CDDL

```
ueid-type = bstr .size (7..33)

ueid-claim = (
     ueid => ueid-type
)
```

3.5. Origination Claim (origination)

TODO: this claim is likely to be dropped in favor of Endorsement identifier and locators.

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"

Table 2

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. origination 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 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. For JSON encoded tokens, this is further base64url encoded.

3.6.1. oemid CDDL

```
oemid-claim = (
    oemid => bstr
)
```

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

The hardware version can be claimed at three different levels, the chip, the circuit board and the final device assembly. An EAT can include any combination these claims.

The hardware version is a simple 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].

The hardware version can also be given by 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 claim is the ASCII text representation of actual digits often printed with a bar code. Use of this claim must comply with the EAN allocation and assignment rules. For example, this requires the manufacturer to obtain a manufacture code from GSI.

Both the simple version string and EAN-13 versions may be included for the same hardware.

```
chip-version-claim = (
    chip-version => tstr
)

chip-version-scheme-claim = (
    chip-version-scheme => $version-scheme
)

board-version-claim = (
    board-version => tstr
)

board-version-scheme-claim = (
    board-version-scheme => $version-scheme
)

device-version-claim = (
    device-version => tstr
)

device-version-scheme-claim = (
    device-version-scheme => $version-scheme
)


ean-type = text .regexp "[0-9]{13}"

ean-chip-version-claim = (
    ean-chip-version => ean-type
)

ean-board-version-claim = (
    ean-board-version => ean-type
)

ean-device-version-claim = (
    ean-device-version => ean-type
)

hardware-version-claims = (
    ? chip-version-claim,
    ? board-version-claim,
    ? device-version-claim,
    ? chip-version-scheme-claim,
    ? board-version-scheme-claim,
    ? device-version-scheme-claim,
    ? ean-chip-version-claim,
    ? ean-board-version-claim,
    ? ean-device-version-claim,
)
```

3.8. Software Description and Version

TODO: Add claims that reference CoSWID.

3.9. The Security Level Claim (security-level)

This claim characterizes the device/entity ability to defend against attacks aimed at capturing the signing key, forging claims and at forging EATs. This is done by defining four security levels as described below. This is similar to the key protection types defined by the Fast Identity Online (FIDO) Alliance [FIDO.Registry].

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 [FIDO.AROE]. 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.

The entity should claim the highest security level it achieves and no higher. This set is not extensible so as to provide a common interoperable description of security level to the relying party. 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 security and its own proprietary claim as a refined indication.

This claim is not intended as a replacement for a proper end-device security certification schemes such as those based on FIPS 140 [FIPS-140] or those based on Common Criteria [Common.Criteria]. The claim made here is solely a self-claim made by the Entity Originator.

3.9.1. security-level CDDL

```
security-level-type = &(
    unrestricted: 1,
    restricted: 2,
    secure-restricted: 3,
    hardware: 4
)

security-level-claim = (
    security-level => security-level-type
)
```

3.10. Secure Boot Claim (secure-boot)

The value of true indicates secure boot is enabled. Secure boot is considered enabled when base software, the firmware and operating system, are under control of the entity manufacturer identified in the oemid claimd described in Section 3.6. This may because the software is in ROM or because it is cryptographically authenticated or some combination of the two or other.

3.10.1. secure-boot CDDL

```
secure-boot-claim = (
    secure-boot => bool
)
```

3.11. Debug Status Claim (debug-status)

This applies to system-wide or submodule-wide debug facilities of the target device / submodule 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-device public keys.

As with all claims, the absence of the debug level claim means it is not reported. A conservative interpretation might assume the Not Disabled state. It could however be that it is reported in a proprietary claim.

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

The higher levels of debug diagnostic information may be useful when debugging a new implementation. Implementations should allow all the target's debug by currently disabled, all other levels require recompile.

There is no enforcement of limits from authorization to a single module or vice versa. There is no assumption, requirement or enforcement of limits from a single module to authorization to a single of submodules. Thus, every submodule must explicitly describe its own debug state. The module or setting query receiving the debug state must be able to parse the debug state of submodules because there is no rules indicating it is assumed of an parent module.

An individual target device / submodule may have multiple debug states. The module or setting query must be able to parse the states from that, or use aggregation or abstraction.

The architecture of some chips or devices may be such that a debug state is not applicable to the entire chip or device. In such a case, the setting automatically forces all submodules to be in the same independently report the status of the whole chip or device device. The module or setting query must be able to parse the state of debug status of the submodules since there is no alternative.

1.11. Enabled

The enabled setting cites manufacturer hardware diagnostics, is currently enabled, then the module must be enabled.

1.12. Debug

The enabled state of debug facilities are currently disabled. It may be possible to enable them in the future, and it may also be possible that they were enabled in the past after the target device/module/submodule/chip/device/chip/device was currently disabled.

1.13. Disabled Due Fault

The fault indicates all debug facilities are currently disabled and have been or since the target device/module/submodule/chip/device/chip/device/chip/device

```
{
  "iss": "https://issuer.example.com",
  "iat": 1658764827,
  "exp": 1658768427,
  "sub_id": {
    "format": "iss_sub",
    "iss": "https://issuer.example.com",
    "sub": "user.12345678901234567890"
  },
  "events": {
    "https://schemas.openid.net/secevent/caep/event-type/session-revoked": {
      "event_timestamp": 1658764827
    }
  }
}
```

```
{
  "iss": "https://issuer.example.com",
  "iat": 1658764827,
  "exp": 1658768427,
  "sub_id": {
    "format": "email",
    "email": "user@example.com"
  },
  "events": {
    "https://schemas.openid.net/secevent/caep/event-type/session-revoked": {
      "event_timestamp": 1658764827
    }
  }
}
```

```
{
  "iss": "https://issuer.example.com",
  "iat": 1658764827,
  "exp": 1658768427,
  "sub_id": {
    "format": "opaque",
    "id": "ZGFlZGJlZWYtMjAyMi0wNzIyLTAwMDAtNDAwMC1iZWVmLTAwMDAwMDAwMDAwMA"
  },
  "events": {
    "https://schemas.openid.net/secevent/caep/event-type/session-revoked": {
      "event_timestamp": 1658764827
    }
  }
}
```

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

```
{
    / nonce /                   10: h'948f886b013a463e8e',
    / UEID /                    11: h'0198f50a4ff6c05861c8860
    / secure-boot /             15: true,
    / debug-disable /           16: 3, / permanent-disable  /
    / timestamp (iat) /          6: 1(1526542894),
    / security-level /          14: 3, / secure restricted OS
    / submods / 20: {
        / first submod, an Android Application /
        "Android App Foo" : {
            / security-level /  14: 1 / unrestricted /
        },

        / 2nd submod, A nested EAT from a secure element /
        "Secure Element EAT" :
            / an embedded EAT, bytes of which are not showm
            h'420123',

        / 3rd submod, information about Linux Android /
        "Linux Android": {
            / security-level /  14: 1 / unrestricted /
        }
    }
}
```

Appendix B. UEID Design Rationale

B.1. Collision Probability

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

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

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

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

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

Table 3

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

Table 4

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

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

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

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

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

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

Table 5

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. 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.

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

- Added UEID design rationale appendix

C.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

C.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

C.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

C.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

C.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

C.7. From draft-ietf-rats-06

- Added boot-seed claim
- Rework CBOR interoperability section
- Added profiles claim and section
- Filled in IANA and other sections for possible preassignment of claim keys for well understood claims

Authors' Addresses

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

Miguel Ballesteros
Qualcomm Technologies Inc.
5775 Morehouse Drive
San Diego, California
United States of America
Phone: +1 858 651 4299
Email:
mballest@qti.qualcomm.com

Laurence Lundblade
Security Theory LLC
Email: lgl@island-resort.com

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

Table of Contents

1. Introduction
   1.1. CWT, JWT and UCCS
   1.2. CDDL
   1.3. Entity Overview
   1.4. EAT Operating Models
   1.5. What is Not Standardized
      1.5.1. Transmission Protocol
      1.5.2. Signing Scheme
2. Terminology
3. The Claims
   3.1. Token ID Claim (cti and jti)
   3.2. Timestamp claim (iat)
   3.3. Nonce Claim (nonce)
      3.3.1. nonce CDDL
   3.4. Universal Entity ID Claim (ueid)
      3.4.1. ueid CDDL
   3.5. Origination Claim (origination)
      3.5.1. origination CDDL
   3.6. OEM Identification by IEEE (oemid)
      3.6.1. oemid CDDL
   3.7. Hardware Version Claims (hardware-version-claims)
   3.8. Software Description and Version
   3.9. The Security Level Claim (security-level)
      3.9.1. security-level CDDL
   3.10. Secure Boot Claim (secure-boot)
      3.10.1. secure-boot CDDL
   3.11. Debug Status Claim (debug-status)
      3.11.1. Enabled
      3.11.2. Disabled
      3.11.3. Disabled Since Boot
      3.11.4. Disabled Permanently
      3.11.5. Disabled Fully and Permanently