Network Working Group
Internet-Draft
Intended status: Informational
Expires: 6 March 2026

S. Frost
Arm Limited
T. Fossati
Linaro
G. Mandyam
Mediatek Inc

2 September 2025

Arm’s Confidential Compute Architecture Reference Attestation Token
draft-ffm-rats-cca-token-02

Abstract

The Arm Confidential Compute Architecture (CCA) is series of hardware and software innovations that enhance Arm’ s support for Confidential Computing for large, compute-intensive workloads. Devices that implement CCA can produce attestation tokens as described in this memo, which are the basis for trustworthiness assessment of the Confidential Compute environment. This document specifies the CCA attestation token structure and semantics.

The CCA attestation token is a profile of the Entity Attestation Token (EAT). This specification describes what claims are used in an attestation token generated by CCA compliant systems, how these claims get serialized to the wire, and how they are cryptographically protected.

This informational document is published as an independent submission to improve interoperability with Arm’s architecture. It is not a standard nor a product of the IETF.

Status of This Memo

Table of Contents

1. Introduction

The Arm Confidential Compute Architecture (CCA) [CCA-ARCH] is a set of hardware [RME] and firmware [RMM] specifications, backed by a reference implementation [TF-RMM] .

CCA provides confidential compute environments, called Realms, that can be dynamically allocated by the Normal world host. The initial state of a Realm, and of the platform on which it executes, can be attested. Attestation allows the Realm owner to establish trust in the Realm, before provisioning any secrets to it. The Realm does not have to inherit the trust from the Non-secure hypervisor which controls it.

As outlined in the RATS Architecture [RFC9334], an Attester produces a signed collection of Claims that constitutes Evidence about its target environment. This document focuses on the output provided by requests from the Realm to the Realm Management Monitor (RMM) management component for an attestation token that covers the state of that Realm and the CCA Platform. This output corresponds to Evidence in [RFC9334] and, as a design decision, the CCA attestation token is a profile of the Entity Attestation Token (EAT) [EAT]. Note that there are other profiles of EAT available, such as [I-D.kdyxy-rats-tdx-eat-profile] and [I-D.mandyam-rats-qwestoken], for use with different use cases and by different attestation technologies.

Since the CCA tokens are consumed by services outside the device, there is an actual need to ensure interoperability. Interoperability needs are addressed here by describing the exact syntax and semantics of the attestation claims, and defining the way these claims are encoded and cryptographically protected.

Further details on concepts expressed below can be found in the Realm Management Monitor specification 1.0 [RMM].

As mentioned in the abstract, this memo documents a vendor extension to the RATS architecture, and is not a standard.

2. Conventions and Definitions

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.

The terms Attester, Relying Party, Verifier, Attestation Result, Target Environment, Attesting Environment and Evidence are defined in [RFC9334]. We use the term "receiver" to refer to Relying Parties and Verifiers.

We use the terms Evidence, "CCA attestation token", and "CCA token" interchangeably. The terms "sender" and Attester are used interchangeably. Likewise, we use the terms Verifier and "verification service" interchangeably.

RoT:
: Root of Trust, the minimal set of software, hardware and data that has to be implicitly trusted in the platform - there is no software or hardware at a deeper level that can verify that the Root of Trust is authentic and unmodified. An example of a RoT suitable for CCA would be an isolated Trusted subsystem responsible for initial measurements, lifecycle state management, identity and attestation services. The services that the RoT provides for securitization of the CCA environment are descibed as Hardware-Enforced Security (HES) - see Section B4.1.5 of [RME].

Realm-World:
: Realm World, provides a security state and physical address range that provides an execution environment for VMs that is isolated from the Normal and Secure worlds. The controlling firmware running in the Realm world can access memory in the Normal world to allow shared buffers. (This is similar to Trusted Execution Environment (TEE), "secure world", or "secure enclave".)

Realm:
: the Realm execution environment, is an Arm CCA environment that can be dynamically allocated by the Normal world Host.

NW-Host:
: Normal world host, refers to the security domain outside of the restricted Root, Secure and Realm worlds. This typically contains the host hypervisor and supervisory services. The NW-Host can allocate and manage resource allocation and can manage the scheduling for other worlds.

In this document, the structure of data is specified in Concise Data Definition Language (CDDL) [RFC8610].

3. CCA Attester Model

There are two kinds of CCA Attester: direct and delegated. Their architectural arrangements are described in Section 3.1 and Section 3.2, respectively.

3.1. Direct

TODO: Issue #16 (https://github.com/SimonFrost-Arm/draft-ffm-rats-cca-token/issues/16)

3.2. Delegated

The structure of the CCA delegated Attester is illustrated in Figure 1. The CCA delegated Attester is a "layered attester" (Section 3.2 of [RFC9334]) with exactly two layers: platform and realm.

The Realm Management Monitor (RMM) is the top layer Attesting Environment. It attests to the initial memory content of each Realm that is executed on a CCA platform, and any dynamic measurements provided by Realm guest code. It uses its own private key called RAK (Realm Attestation Key) to sign the claims regarding the requesting Realm.

The HES (Hardware Enforced Security) is the bottom layer Attesting Environment, which acts as the CCA platform hardware RoT. It attests to the executables and configuration contents of the "Monitor Security Domain", which includes the RMM, as well as a few relevant CCA parameters (e.g., the CCA platform implementation identifier), and the security lifecycle state of the platform. Additionally, it generates the RAK keypair, transfers it over a trusted channel to the RMM, and stores the hash of the RAK public key in a claim that is signed using the CCA Platform Attestation Key (CPAK) as part of the platform Evidence.

The CCA Evidence produced in delegated mode comprises two separately signed EATs, one for the platform, another for the realm, wrapped in a CMW [CMW] collection. The intra-collection binding is detailed in Section 4.10.

```
                                      .----------.
                                      |          |
                                      | Verifier |
                                      |          |
                                      '----------'
                                           ^
   .-----------------------------------.   |
   |    .-------------.                |   |
   |    | Target      |                |   |
   |    | Environment +----.           |   | Layered
   |    |             | TE |           |   | Evidence for
   |    | Realm 1     |    +----.      |   | Platform and
   |    '-+--+--------'    | TE |      |   | Realm
   |      |  | Realm 2     |    |      |   |
   |      |  '-+--+--------'    |      |   |
   |      |    |  |  ...        |      |   |
   |      |    |  '-+-----------'      |   |
   |      |    |    | Collect Claims   |   |
   | .----| ---|----|----------------. |   |
   | |    v    v    v    Target      | |   |
   | |   .-----------.   Environment | |   |
   | |  | Realm       |              | |   |
   | |  | Management  +-------------------'
   | |  | Monitor     | .----------. | |
   | |  | (RMM)       | |   BLs    | | |
   | |  |             | '----------' | |
   | |  |             |              | |
   | |  | Attesting   | .----------. | |
   | |  | Environment | |   CFGs   | | |
   | |   '-----------'  '----------' | |
   | |    ^                 Platform | |
   | '----|--------------+-----------' |
   |      | Evidence     | Collect     |
   |      | for          | Claims      |
   |      | Platform     v             |
   |  .---+----------------------.     |
   | | Hardware                   |    |
   | | Enforced                   |    |
   | | Security      Attesting   |    |
   | | (HES)         Environment |    |
   |  '--------------------------'     |
   '-----------------------------------'
```

Figure 1: CCA Attester

3.3. Boot Phase

The HES Attesting Environment is responsible for collecting the information to be represented in CCA platform claims and to assemble them into Evidence.

The Main Bootloader, executing at boot-time, measures the trusted computing base (TCB) of the Realm World - i.e., loaded firmware components and the associated configuration payloads - and sends them to the HES RoT to be stored isolated. See Figure 2.

```
    i-th Target    Main Boot        HES
    Environment      Loader         RoT
          |             |             |
.---------|-------------|-------------|----.
| loop i  |             |             |    |
|         | measure     |             |    |
|         |o------------+             |    |
|         |             | write       |    |
|         |             | measurement |    |
|         |             +------------>|    |
'---------|-------------|-------------|----'
          |             |             |
```

Figure 2: CCA Attester Boot Phase

3.4. Run-time Phase

The Realm Management Monitor (RMM), executing at run-time, maintains measurements for the state of a Realm. It can respond to requests issued from a Realm for an attestation token relevant for that Realm by obtaining a CCA Platform attestation token from the HES RoT and combining that with an attestation token containing Evidence reflecting Realm state.

The HES RoT, executing at run-time, maintains measurements for the state of the CCA platform TCB, including the lifecycle state of the CCA platform. It can answer requests coming from the RMM to collect and format claims corresponding to that state and use a CCA Platform Attestation Key (CPAK) to sign them (see Figure 3). How the CPAK is derived is implementation-specific.

```
              HES           Realm
              RoT           Manager     Verifier
               |               |           |
               |               |           |
     Platform  |               |           |
     Boot      |               |           |
     State     | Req. Platform |           |
       |       | Token (#RAK)  |           |
       |       |<--------------+           |
       v       |               |           |
     Platform  |               |           |
     Token     |               |           |
               |               |           |
      sign .---+               |           |
      w/   |   |               |           |
      CPAK '-->|   Plat Token  |           |
               +-------------->|           |
               |               |           |
               |     Realm     |           |
               |     State     |           |
               |       |       |           |
               |       |       |           |
               |       v       |           |
               |     Realm     |           |
               |     Token     |           |
               |               |           |
               |      sign .---+           |
               |      w/   |   |           |
               |      RAK  '-->|           |
               |               |           |
               |               |           |
               |               | CCA Token |
               |               +---------->|
               |               |           |
```

Figure 3: CCA Attester Run-time Phase

A reference implementation of the CCA Attester is provided by [TF-RMM].

4. CCA Claims

This section describes the claims to be used in a CCA reference attestation token.

There are two logical sections within the CCA attestation token, relating to the two Target Environment elements:

* The CCA Platform token

* The Realm state token

The two sections use inter-related claims to bind together into a single logical unit. See Section 9 for more details.

The above tokens are presented to the requester within a top level Conceptual Message Wrapper (CMW) collection [CMW].

CDDL [RFC8610] along with text descriptions is used to define each claim independent of encoding. The following CDDL type(s) are reused by different claims:

```
arm-platform-hash-type = bytes .size 32 /
                         bytes .size 48 /
                         bytes .size 64
```

Two conventions are used to encode the Right-Hand-Side (RHS) of a claim: the postfix -label is used for EAT-defined claims, and the postfix -key for PSA-originated claims.

4.1. CCA Attestation Token top level wrapper

The above tokens are presented to the requester within a top level CMW collection [CMW]. The collection map has two entries, one for a bstr encoding of the CCA Platform token and the other for a bstr encoding of the Realm state token/

```
; CMW (draft-ietf-rats-msg-wrap) Collection
cca-token = #6.907(cca-token-cmw)

; CoAP Content-Format for "application/eat+cwt"
eat-cwt-coap-cf = 263

cca-token-cmw = {
  0xACCA => [
    eat-cwt-coap-cf
    bytes .cbor COSE_Sign1<arm-platform-claims>
  ]
  0xACD1 => [
    eat-cwt-coap-cf
    bytes .cbor COSE_Sign1<cca-realm-claims>
  ]
}
```

4.2. CCA Platform token Claims

4.3. Caller Claims

4.3.1. CCA Platform Nonce

The Nonce claim is used to carry a challenge provided by the caller to demonstrate freshness of the generated token.

The EAT [EAT] nonce (claim key 10) is used. Since the EAT nonce claim offers flexiblity for different attestation technologies, this specifications applies the following constraints to the nonce-type:

* The length MUST be either 32, 48, or 64 bytes.

* Only a single nonce value is conveyed. The array notation MUST NOT be used for encoding the nonce value.

Where the CCA Platform implementation uses the Delegated Token signing model Section 4.10, the value of the Nonce claim will be a hash of the Realm Public Key claim of the CCA Realm State token Section 4.8.7.

This claim MUST be present in a CCA Platform attestation token.

```
arm-platform-challenge-label = 10

arm-platform-challenge = (
    arm-platform-challenge-label => arm-platform-hash-type
)
```

4.4. Target Identification Claims

4.4.1. CCA Platform Instance ID

The Instance ID claim represents the unique identifier of the Platform Attestation Key (PAK). The EAT ueid (claim key 256) of type RAND is used. The following constraints apply to the ueid-type:

* The length MUST be 33 bytes.

* The first byte MUST be 0x01 (RAND) followed by the 32-byte unique identifier of the PAK.

```
eat-ueid-rand-type = bytes .join eat-ueid-rand-fmt

eat-ueid-rand-fmt = [
  ; the type byte is 0x01
  ueid-rand-typ
  bytes .size 32
]

ueid-rand-typ = h'01'
```

This claim MUST be present in a CCA Platform attestation token.

```
arm-platform-instance-id-label = 256 ; EAT ueid

arm-platform-instance-id-type = eat-ueid-rand-type

arm-platform-instance-id = (
    arm-platform-instance-id-label => arm-platform-instance-id-type
)
```

4.4.2. CCA Platform Implementation ID

The Implementation ID claim uniquely identifies the implementation of the CCA Platform. A verification service uses this claim to locate the details of the CCA Platform implementation from an Endorser or manufacturer. Such details are used by a verification service to determine the security properties or certification status of the CCA Platform implementation.

The value and format of the ID is decided by the manufacturer or a particular certification scheme. For example, the ID could take the form of a product serial number, database ID, or other appropriate identifier.

This claim MUST be present in a CCA Platform attestation token.

Note that this identifies the CCA Platform implementation, not a particular instance. To uniquely identify an instance, see the Instance ID claim Section 4.4.1.

```
arm-platform-implementation-id-label = 2396 ; PSA implementation ID
arm-platform-implementation-id-type = bytes .size 32

arm-platform-implementation-id = (
    arm-platform-implementation-id-label =>
        arm-platform-implementation-id-type
)
```

4.5. Target State Claims

4.5.1. CCA Platform Profile Definition

The CCA platform profile claim identifies the EAT profile to which the CCA platform token conforms. This allows a receiver to assign the intended semantics to the rest of the claims found in the token.

The EAT eat_profile (claim key 265) is used.

The format of the CCA platform profile claim is defined as a text string of value "tag:arm.com,2023:cca_platform#1.0.0".

This claim MUST be present in a CCA Platform attestation token.

See Section 4.9, for considerations about backwards compatibility with previous versions of the CCA Platform attestation token format.

```
arm-platform-profile-label = 265 ; EAT profile

arm-platform-profile-type = "tag:arm.com,2023:cca_platform#1.0.0"

arm-platform-profile = (
    arm-platform-profile-label => arm-platform-profile-type
)
```

4.5.2. Security Lifecycle

The Security Lifecycle claim represents the current lifecycle state of the CCA Platform.

The state is represented by an integer that is divided as follows:

* major[15:8] - CCA Platform security lifecycle state, and

* minor[7:0] - IMPLEMENTATION DEFINED state.

The CCA Platform lifecycle states are illustrated in Figure 4. A non debugged CCA platform will be in psa-lifecycle-secured state. Realm Management Security Domain debug is always recoverable, and would therefore be represented by psa-lifecycle-non-psa-rot-debug state. Root world debug is recoverable on a HES system and would be represented by psa-lifecycle-recoverable-psa-rot state. On a non-HES system Root world debug is usually non-recoverable, and would be represented by psa-lifecycle-lifecycle-decommissioned state

This claim MUST be present in a CCA Platform attestation token.

```
                    .--------------------------.
                    | Device Assembly and Test |
                    '------------+-------------'
                                 | Device
                                 | Lockdown
                                 v
                   .----------------------------.
                   | CCA Security Provisioning  |
                   '-----------+----------------'
                               |
                  Provisioning |  .---------------------.
                    Lockdown   |  |                     |
                               v  v                     |
                         .----------------.             |Recoverable
          .--------------+    Secured     +--------.    |
          |              '-+--------------'        |    |
         Non               |          ^     Recoverable |
     Recoverable       RM Debug       |     Root Debug  |
     Root Debug          Enable       |            |    |
          |                |          |            |    |
          |                v          |            v    |
          |              .---------- -+--.      .-------+-.
          |              | Realm Manager |      |  Root   |
          |              |    Debug      |      | Debug   |
          |              '---------------'      '--+------'
          |                                        |
          |            .---------------.           |
          '----------->+   Terminate   +<----------'
                       '---------------'
                               |
                               |
                               v
                       .----------------.
                       | Decommissioned |
                       '----------------'
```

Figure 4: CCA Platform Lifecycle States

The CDDL representation is shown below. Table 1 provides the mappings between Figure 4 and the data model.

```
arm-platform-lifecycle-label = 2395 ; PSA lifecycle

arm-platform-lifecycle-unknown-type = 0x0000..0x00ff
arm-platform-lifecycle-assembly-and-test-type = 0x1000..0x10ff
arm-platform-lifecycle-arm-platform-rot-provisioning-type = 0x2000..0x20ff
arm-platform-lifecycle-secured-type = 0x3000..0x30ff
arm-platform-lifecycle-non-arm-platform-rot-debug-type = 0x4000..0x40ff
arm-platform-lifecycle-recoverable-arm-platform-rot-debug-type = 0x5000..0x50ff
arm-platform-lifecycle-decommissioned-type = 0x6000..0x60ff

arm-platform-lifecycle-type =
    arm-platform-lifecycle-unknown-type /
    arm-platform-lifecycle-assembly-and-test-type /
    arm-platform-lifecycle-arm-platform-rot-provisioning-type /
    arm-platform-lifecycle-secured-type /
    arm-platform-lifecycle-non-arm-platform-rot-debug-type /
    arm-platform-lifecycle-recoverable-arm-platform-rot-debug-type /
    arm-platform-lifecycle-decommissioned-type

arm-platform-lifecycle = (
    arm-platform-lifecycle-label => arm-platform-lifecycle-type
)
```

psa-lifecycle-unknown-type is not shown in Figure 4; it represents an invalid state that must not occur in a system.

CDDL	Lifecycle States
psa-lifecycle-unknown-type	
psa-lifecycle-assembly-and-test-type	Assembly and Test
psa-lifecycle-psa-rot-provisioning-type	CCA Platform Provisioning
psa-lifecycle-secured-type	Secured
psa-lifecycle-non-psa-rot-debug-type	Non-Recoverable CCA Platform Debug
psa-lifecycle-recoverable-psa-rot-debug-type	Recoverable CCA Platform Debug
psa-lifecycle-decommissioned-type	Decommissioned

Table 1: Lifecycle States Mappings

4.5.3. Platform Config

The CCA platform config claim describes the set of chosen implementation options of the CCA platform. As an example, these may include a description of the level of physical memory protection which is provided.

The CCA platform config claim is expected to contain the System Properties field which is present in the Root Non-volatile Storage (RNVS) public parameters.

This claim MUST be present in a CCA Platform attestation token.

```
arm-platform-config-label = 2401 ; PSA platform range
                                 ; TBD: add to IANA registration
arm-platform-config-type = bytes

arm-platform-config = (
    arm-platform-config-label => arm-platform-config-type
)
```

4.6. Software Inventory Claims

4.6.1. Software Components

The Software Components claim is a list of software components which can affect the behavior of the CCA platform.

This claim MUST be present in a CCA Platform attestation token.

Each entry in the Software Components list describes one software component using the attributes described in the following subsections. Unless explicitly stated, the presence of an attribute is OPTIONAL.

Note that, as described in [RFC9334], a relying party will typically see the result of the appraisal process from the Verifier in form of an Attestation Result, rather than the CCA Platform token from the attesting endpoint. Therefore, a relying party is not expected to understand the Software Components claim. Instead, it is for the Verifier to check this claim against the available Reference Values and provide an answer in form of an "high level" Attestation Result, which may or may not include the original Software Components claim.

```
arm-platform-sw-components-label = 2399 ; PSA software components

arm-platform-sw-component = {
  ? 1 => text,                   ; component type
    2 => arm-platform-hash-type, ; measurement value
  ? 4 => text,                   ; version
    5 => arm-platform-hash-type, ; signer id
  ? 6 => text,                   ; hash algorithm identifier
}

arm-platform-sw-components = (
    arm-platform-sw-components-label => [ + arm-platform-sw-component ]
)
```

4.6.1.1. Component Type

The Component Type attribute (key=1) is a short string representing the role of this software component. This attribute is intended for use as a hint to help the verifier understand how to evaluate the CCA platform software component measurement value.

This attribute is optional in a CCA Platform software component.

4.6.1.2. Measurement Value

The Measurement Value attribute (key=2) represents a hash of the invariant software component in memory at the time it was initialized. The value MUST be a cryptographic hash of 256 bits or stronger.

This attribute MUST be present in a PSA software component.

4.6.1.3. Version

The Version attribute (key=4) is the issued software version in the form of a text string. The meaning of this string is defined by the software component vendor.

This attribute is optional in a CCA Platform software component.

4.6.1.4. Signer ID

The Signer ID attribute (key=5) uniquely identifies the signer of the software component. The identification is typically accomplished by hashing the signer’s public key. The value of this attribute will correspond to the entry in the original manifest for the component. This can be used by a Verifier to ensure the components were signed by an expected trusted source.

This attribute MUST be present in a CCA Platform software component.

4.6.1.5. Measurement Description

The Measurement Description attribute (key=6) contains a string identifying the hash algorithm used to compute the corresponding Measurement Value. The string SHOULD be encoded according to "Hash Name String" in the "Named Information Hash Algorithm Registry" [IANA.named-information].

4.7. Verification Claims

The following claims are part of the CCA Platform token (and therefore still Evidence) but aim to help receivers, including relying parties, with the processing of the received attestation Evidence.

4.7.1. Verification Service Indicator

The Verification Service Indicator claim is a hint used by a relying party to locate a verification service for the token. The value is a text string that can be used to locate the service (typically, a URL specifying the address of the verification service API). A Relying Party may choose to ignore this claim in favor of other information.

```
; PSA verification service
arm-platform-verification-service-label = 2400
arm-platform-verification-service-type = text

arm-platform-verification-service = (
    arm-platform-verification-service-label =>
        arm-platform-verification-service-type
)
```

It is assumed that the relying party is pre-configured with a list of trusted verification services and that the contents of this hint can be used to look up the correct one. Under no circumstances must the relying party be tricked into contacting an unknown and untrusted verification service since the returned Attestation Result cannot be relied on.

Note: This hint requires the relying party to parse the content of the CCA Platform token. Since the relying party may not be in possession of a trust anchor to verify the digital signature, it uses the hint in the same way as it would treat any other information provided by an external party, which includes attacker-provided data.

The CCA platform verification service indicator claim is OPTIONAL in a CCA platform token.

4.7.2. CCA Platform Hash Algorithm ID

The CCA platform hash algorithm ID claim is a text string that identifies the algorithm used to calculate the extended measurements in the CCA platform token.

The string SHOULD be encoded according to "Hash Name String" in the "Named Information Hash Algorithm Registry" [IANA.named-information].

The CCA platform hash algorithm ID claim MUST be present in a CCA platform token.

```
arm-platform-hash-algo-id-label = 2402 ; PSA platform range
                                       ; TBD: add to IANA registration

arm-platform-hash-algo-id = (
    arm-platform-hash-algo-id-label => text
)
```

4.8. CCA Realm state token Claims

The CCA Realm state token contains claims that represent the Target Environment that is the Realm that requested the attestation report.

4.8.1. Realm Nonce

The Nonce claim is used to carry a challenge provided by the caller to demonstrate freshness of the generated token.

The EAT [EAT] nonce (claim key 10) is used. Since the EAT nonce claim offers flexiblity for different attestation technologies, this specification applies the following constraints to the nonce-type:

* The length MUST be 64 bytes.

* Only a single nonce value is conveyed. The array notation MUST NOT be used for encoding the nonce value.

This claim MUST be present in a CCA Realm state attestation token.

```
cca-realm-challenge-label = 10
cca-realm-challenge-type = bytes .size 64

cca-realm-challenge = (
    cca-realm-challenge-label => cca-realm-challenge-type
)
```

4.8.2. CCA Platform Profile Definition

The Realm profile claim identifies the EAT profile to which the Realm token conforms. This allows a receiver to assign the intended semantics to the rest of the claims found in the token.

The EAT eat_profile (claim key 265) is used.

The format of the CCA platform profile claim is defined as a text string of value "tag:arm.com,2023:realm#1.0.0".

This claim is OPTIONAL in a CCA Realm attestation token. If the Realm profile is not included in a CCA Realm token then the profile value used in the CCA Platform token should refer to a profile that describes both Platform and Realm claims.

```
cca-realm-profile-label = 265 ; EAT profile

cca-realm-profile-type = "tag:arm.com,2023:realm#1.0.0"

cca-realm-profile = (
    cca-realm-profile-label => cca-realm-profile-type
)
```

4.8.3. Realm Personalisation Value

The Realm Personalization Value (RPV) claim contains the RPV which was provided at Realm creation.

This claim MUST be present in a CCA Realm state attestation token.

```
cca-realm-personalization-value-label = 44235
cca-realm-personalization-value-type = bytes .size 64

cca-realm-personalization-value = (
    cca-realm-personalization-value-label =>
        cca-realm-personalization-value-type
)
```

4.8.4. Realm Initial Measurement

The Realm Initial Measurement claim contains the compound extension of measurements taken of Realm memory and state before the Realm is activated.

This claim MUST be present in a CCA Realm state attestation token.

```
cca-realm-initial-measurement-label = 44238

cca-realm-initial-measurement = (
    cca-realm-initial-measurement-label => cca-realm-measurement-type
)
```

4.8.5. Realm Extensible Measurements

The Realm Extensible Measurements claim contains measurements provided by Realm guest software and extended to the set of Realm Extensible Measurements maintained by the RMM.

This claim MUST be present in a CCA Realm state attestation token.

```
cca-realm-extensible-measurements-label = 44239

cca-realm-extensible-measurements = (
    cca-realm-extensible-measurements-label =>
        [ 4*4 cca-realm-measurement-type ]
)
```

4.8.6. Realm Hash Algorithm Measurements

The Realm hash algorithm ID claim identifies the algorithm used to calculate all hash values which are present in the Realm token.

The string value of the claim SHOULD be encoded according to "Hash Name String" in the "Named Information Hash Algorithm Registry" [IANA.named-information].

This claim MUST be present in a CCA Realm state attestation token.

```
cca-realm-hash-algo-id-label = 44236

cca-realm-hash-algo-id = (
    cca-realm-hash-algo-id-label => text
)
```

4.8.7. Realm Public Key

The Realm public key claim identifies the attestation key which is used to sign the Realm token

The value of the Realm public key claim is a byte string representation of a COSE_Key.

This claim MUST be present in a CCA Realm state attestation token.

```
cca-realm-public-key-label = 44237

; See RFC8152 for definition of COSE_Key
cca-realm-public-key-type = bstr .cbor COSE_Key

cca-realm-public-key = (
    cca-realm-public-key-label => cca-realm-public-key-type
)
```

4.8.8. Realm Public Key Hash Algorithm ID

The Realm public key hash algorithm identifier claim identifies the algorithm used to hash the value of the Realm Public Key claim Section 4.8.7 such that it can be presented as a Challenge for the bound CCA Platform token Section 4.10.

This claim MUST be present in a CCA Realm state attestation token.

```
cca-realm-public-key-hash-algo-id-label = 44240

cca-realm-public-key-hash-algo-id = (
    cca-realm-public-key-hash-algo-id-label => text
)
```

4.9. Backwards Compatibility Considerations

This profile conforms to the claims in the Beta2 release of the 1.0 release of the Realm Management Monitor specification. [RMM]. There has not been a prior release of this specification to the 1.0 release. Hence this section is a place holder for claim changes introduced in future releases.

4.10. Token Binding

The reference implementation uses a 'Delegated Model' for token signing. In this model, the completion of signing operations for the CCA token is delegated from the CCA Platform RoT to the RMM. When the RMM initialises, it obtains a 'Realm Attestation Token' (RAK) signing key pair from the CCA Platform RoT. The public part of that key pair is hashed and used as a challenge to obtain a CCA Platform token (signed by the CCA Platform RoT). When guest code in a Realm requests a CCA Attestation token, the RMM prepares a Realm state token, signed by the RAK private key, then wraps both tokens in a CMW Collection. The two tokens are bound together by the Nonce claim in the CCA Platform token having the same value as a hash of the Realm Public key claim in the Realm state token (using the hash algorithm identified by the Realm Public Key Hash Algorithm ID claim).

A verifier MUST check this binding is valid when verifying a CCA Attestation token.

An implementation may choose instead a 'Direct Model'. In this model, when guest code in a Realm requests a CCA Attestation token, the RMM prepares a Realm state claim set, but does not wrap it in a CMW. Instead, the claim set is hashed and this value is used as a Challenge to obtain a CCA Platform token, signed by the CCA Platform RoT. The CCA Platform and Realm state claim set are presented within a CMW Collection as in the Delegated model. The two parts of the collection are bound together by the Nonce claim in the CCA Platform token having the same value as the hash of the Realm state claim set. If the Direct Model is used, the CCA Platform profile claim Section 4.5.1 MUST have a different value from the reference profile. The map value within the CCA Attestation token CMW Collection for the Realm state claim set MUST also have a different value to that used for a Realm state CMW token. In such a profile, the Realm Public Key Section 4.8.7 and Realm Public Key Hash Algorithm ID Section 4.8.8 claims will not be used.

4.11. Reference Profile

4.11.1. Token Encoding and Signing

The CCA attestation token is encoded in CBOR [STD94] format. The CBOR representation of a CCA attestation token MUST be "valid" according to the definition in Section 1.2 of [STD94]. Besides, only definite-length string, arrays, and maps are allowed.

Given that a PSA Attester is typically found in a constrained device, it MAY NOT emit CBOR preferred serializations (Section 4.1 of [STD94]). Therefore, the Verifier MUST be a variation-tolerant CBOR decoder. TODO: Issue #31 (https://github.com/SimonFrost-Arm/draft-ffm-rats-cca-token/issues/31) need different narrative from IoT reasons

Cryptographic protection is obtained by wrapping the CCA Platform and Realm state claims-set in a COSE Web Token (CWT) [RFC8392]. The signature structure MUST be a tagged (18) COSE_Sign1 [STD96].

Acknowledging the variety of markets, regulations and use cases in which the CCA attestation token can be used, the baseline profile does not impose any strong requirement on the cryptographic algorithms that need to be supported by Attesters and Verifiers. The flexibility provided by the COSE format should be sufficient to deal with the level of cryptographic agility needed to adapt to specific use cases. It is RECOMMENDED that commonly adopted algorithms are used, such as those discussed in [COSE-ALGS]. It is expected that receivers will accept a wider range of algorithms, while Attesters would produce CCA tokens using only one such algorithm.

The CCA Platform token is always directly signed by the CCA Platform RoT. Therefore, the CCA claims-set is never carried in a Detached EAT bundle (Section 5 of [EAT]).

4.11.2. Freshness Model

The CCA token supports the freshness models for attestation Evidence based on nonces and epoch handles (Section 10.2 and Section 10.3 of [RFC9334]) using the nonce claim to convey the nonce or epoch handle supplied by the Verifier. No further assumption on the specific remote attestation protocol is made.

Note that use of epoch handles is constrained by the type restrictions imposed by the eat_nonce syntax. For use in CCA tokens, it must be possible to encode the epoch handle as an opaque binary string between 8 and 64 octets.

4.11.3. Synopsis

Table 2 presents a concise view of the requirements described in the preceding sections.

Issue	Profile Definition
CBOR/JSON	CBOR MUST be used
CBOR Encoding	Definite length maps and arrays MUST be used
CBOR Encoding	Definite length strings MUST be used
CBOR Serialization	Variant serialization MAY be used
COSE Protection	COSE_Sign1 MUST be used
Algorithms	[COSE-ALGS] SHOULD be used
Detached EAT Bundle Usage	Detached EAT bundles MUST NOT be sent
Verification Key Identification	Any identification method listed in Appendix F.1 of [EAT]
Endorsements	See Section 7.2
Freshness	nonce or epoch ID based
Claims	Those defined in Section 4. As per general EAT rules, the receiver MUST NOT error out on claims it does not understand.

Table 2: Baseline Profile

5. Collated CDDL

```
=============== NOTE: '\' line wrapping per RFC 8792 ================

cca-token = #6.907(cca-token-cmw)
eat-cwt-coap-cf = 263
cca-token-cmw = {
  0xACCA => [
        eat-cwt-coap-cf,
        bytes .cbor COSE_Sign1<arm-platform-claims>,
],
  0xACD1 => [
        eat-cwt-coap-cf,
        bytes .cbor COSE_Sign1<cca-realm-claims>,
```

```
   ],
   }
   COSE_Sign1<C> = #6.18([
         Headers,
         payload: bytes .cbor C,
         signature: bytes,
   ])
   cca-realm-claims = cca-realm-claim-map
   cca-realm-claim-map = {
     cca-realm-challenge,
     ? cca-realm-profile,
     cca-realm-personalization-value,
     cca-realm-initial-measurement,
     cca-realm-extensible-measurements,
     cca-realm-hash-algo-id,
     cca-realm-public-key,
     cca-realm-public-key-hash-algo-id,
     cca-realm-mec-policy,
   }
   cca-realm-challenge-label = 10
   cca-realm-challenge-type = bytes .size 64
   cca-realm-challenge = (cca-realm-challenge-label => cca-realm-\
                                                      challenge-type)
   cca-realm-extensible-measurements-label = 44239
   cca-realm-extensible-measurements = (cca-realm-extensible-\
                  measurements-label => [4*4cca-realm-measurement-type])
   cca-realm-hash-algo-id-label = 44236
   cca-realm-hash-algo-id = (cca-realm-hash-algo-id-label => text)
   cca-realm-initial-measurement-label = 44238
   cca-realm-initial-measurement = (cca-realm-initial-measurement-\
                                     label => cca-realm-measurement-type)
   cca-realm-measurement-type = bytes .size 32 / bytes .size 48 / \
                                                       bytes .size 64
   cca-realm-personalization-value-label = 44235
   cca-realm-personalization-value-type = bytes .size 64
   cca-realm-personalization-value = (cca-realm-personalization-value-\
                          label => cca-realm-personalization-value-type)
   cca-realm-profile-label = 265
   cca-realm-profile-type = "tag:arm.com,2023:realm#1.0.0"
   cca-realm-profile = (cca-realm-profile-label => cca-realm-profile-\
                                                                type)
   cca-realm-public-key-hash-algo-id-label = 44240
   cca-realm-public-key-hash-algo-id = (cca-realm-public-key-hash-algo-\
                                                    id-label => text)
   cca-realm-public-key-label = 44237
   cca-realm-public-key-type = bstr .cbor COSE_Key
   cca-realm-public-key = (cca-realm-public-key-label => cca-realm-\
                                                     public-key-type)
```

```
   cca-realm-mec-policy-label = 44243
   cca-realm-mec-policy = (cca-realm-mec-policy-label => "shared" / "\
                                                            private")
   label = int / tstr
   values = any
   COSE_Key = {
     1 => tstr / int,
     ? 2 => bstr,
     ? 3 => tstr / int,
     ? 4 => [+ tstr / int],
     ? 5 => bstr,
     * label => values,
   }
   arm-platform-claims = arm-platform-claim-map
   arm-platform-claim-map = {
     arm-platform-profile,
     arm-platform-challenge,
     arm-platform-implementation-id,
     arm-platform-instance-id,
     arm-platform-config,
     arm-platform-lifecycle,
     arm-platform-sw-components,
     ? arm-platform-verification-service,
     arm-platform-hash-algo-id,
   }
   arm-platform-challenge-label = 10
   arm-platform-challenge = (arm-platform-challenge-label => arm-\
                                                  platform-hash-type)
   arm-platform-config-label = 2401
   arm-platform-config-type = bytes
   arm-platform-config = (arm-platform-config-label => arm-platform-\
                                                         config-type)
   arm-platform-hash-algo-id-label = 2402
   arm-platform-hash-algo-id = (arm-platform-hash-algo-id-label => text)
   arm-platform-hash-type = bytes .size 32 / bytes .size 48 / bytes .\
                                                              size 64
   arm-platform-implementation-id-label = 2396
   arm-platform-implementation-id-type = bytes .size 32
   arm-platform-implementation-id = (arm-platform-implementation-id-\
                        label => arm-platform-implementation-id-type)
   arm-platform-instance-id-label = 256
   arm-platform-instance-id-type = eat-ueid-rand-type
   arm-platform-instance-id = (arm-platform-instance-id-label => arm-\
                                            platform-instance-id-type)
   arm-platform-profile-label = 265
   arm-platform-profile-type = "tag:arm.com,2023:cca_platform#1.0.0"
   arm-platform-profile = (arm-platform-profile-label => arm-platform-\
                                                        profile-type)
```

```
   arm-platform-lifecycle-label = 2395
   arm-platform-lifecycle-unknown-type = 0x0000 .. 0x00ff
   arm-platform-lifecycle-assembly-and-test-type = 0x1000 .. 0x10ff
   arm-platform-lifecycle-arm-platform-rot-provisioning-type = 0x2000 .\
                                                                . 0x20ff
   arm-platform-lifecycle-secured-type = 0x3000 .. 0x30ff
   arm-platform-lifecycle-non-arm-platform-rot-debug-type = 0x4000 .. \
                                                                  0x40ff
   arm-platform-lifecycle-recoverable-arm-platform-rot-debug-type = \
                                                          0x5000 .. 0x50ff
   arm-platform-lifecycle-decommissioned-type = 0x6000 .. 0x60ff
   arm-platform-lifecycle-type = arm-platform-lifecycle-unknown-type / \
   arm-platform-lifecycle-assembly-and-test-type / arm-platform-\
   lifecycle-arm-platform-rot-provisioning-type / arm-platform-\
   lifecycle-secured-type / arm-platform-lifecycle-non-arm-platform-rot\
   -debug-type / arm-platform-lifecycle-recoverable-arm-platform-rot-\
                 debug-type / arm-platform-lifecycle-decommissioned-type
   arm-platform-lifecycle = (arm-platform-lifecycle-label => arm-\
                                              platform-lifecycle-type)
   arm-platform-sw-components-label = 2399
   arm-platform-sw-component = {
     ? 1 => text,
     2 => arm-platform-hash-type,
     ? 4 => text,
     5 => arm-platform-hash-type,
     ? 6 => text,
   }
   arm-platform-sw-components = (arm-platform-sw-components-label => [\
                                         + arm-platform-sw-component])
   arm-platform-verification-service-label = 2400
   arm-platform-verification-service-type = text
   arm-platform-verification-service = (arm-platform-verification-\
                service-label => arm-platform-verification-service-type)
   eat-ueid-rand-type = bytes .join eat-ueid-rand-fmt
   eat-ueid-rand-fmt = [
     ueid-rand-typ,
     bytes .size 32,
   ]
   ueid-rand-typ = h'01'
   Headers = (
     protected: empty_or_serialized_map,
     unprotected: header_map,
     )
   empty_or_serialized_map = bstr .cbor header_map / bstr .size 0
   header_map = {
     Generic_Headers,
     * label => values,
   }
```

```
Generic_Headers = (
  ? 1 => int / tstr,
  ? 2 => [+ label],
  ? 3 => tstr / int,
  ? 4 => bstr,
  ? (5 => bstr // 6 => bstr),
  )
```

6. Signing key implementation alternatives

In the CCA Platform reference design, PAKs (Section 3.4, Paragraph 2) are raw public keys.

Some implementations may choose to use a PAK that is a certified public key. If this option is taken, the value of the CCA Platform Profile Definition claim Section 4.5.1 MUST be altered from the reference implementation value.

TODO: Issue #32 (https://github.com/SimonFrost-Arm/draft-ffm-rats-cca-token/issues/32) Cut the following block?

Certified public keys require the manufacturer to run the certification authority (CA) that issues X.509 certs for the PAKs. (Note that operating a CA is a complex and expensive task that may be unaffordable to certain manufacturers.)

Using certified public keys offers better scalability properties when compared to using raw public keys, namely:

* storage requirements for the Verifier are minimised - the same manufacturer’s trust anchor is used for any number of devices,

* the provisioning model is simpler and more robust since there is no need to notify the Verifier about each newly manufactured device,

Furthermore, existing and well-understood revocation mechanisms can be readily used.

TODO: Issue #35 (https://github.com/SimonFrost-Arm/draft-ffm-rats-cca-token/issues/35) improve cert description

The PAK’s X.509 cert can be inlined in the CCA Platform token using the x5chain COSE header parameter [COSE-X509] at the cost of an increase in the CCA Platform token size. Note that the exact split between pre-provisioned and inlined certs may vary depending on the specific deployment. In that respect, x5chain is quite flexible: it can contain the end-entity (EE) cert only, the EE and a partial chain, or the EE and the full chain up to the trust anchor (see Section 2 of [COSE-X509] for the details).

TODO: Issue #33 (https://github.com/SimonFrost-Arm/draft-ffm-rats-cca-token/issues/33) lose following as IoT centric??

Constraints around network bandwidth and computing resources available to endpoints, such as network buffers, may dictate a reasonable split point.

7. CCA Attestation Token Verification

To verify the token for the reference profile, the initial need is to check correct encoding for the token. Primary trust is established by checking the signing of the CCA Platform token CWT. The key used for verification is supplied to the Verifier by an authorized Endorser along with the corresponding Attester’s Instance ID. For the verifier, the CCA Platform Instance ID Section 4.4.1 claim is used to assist locating the key used to verify the signature covering the CCA Platform CWT token. The verifier can also be supplied with the information that the key instance has been revoked and is no longer valid.

Additional validation checks on the token are:

* Checking that the binding between the CCA Platform token and the Realm state token is valid Section 4.10}. This has the side effect of establishing the trustworthiness of the RAK public key.

* Validating that the Realm state token is correctly signed by the RAK.

* Checking that the value of the lll claim is psa-lifecycle-secured state. Note that some other values of this claim (psa-lifecycle-non-psa-rot-debug and psa-lifecycle-recoverable-psa-rot states) may indicate that the attester is only temporarily unsuitable and the verifier may choose the to indicate this as a contraindication rather than a full verification failure. See discussion of the CCA platform lifecycle in [RMM].

The Verifier will typically operate a policy where values of some of the claims in this profile can be compared to reference values, registered with the Verifier for a given deployment, in order to confirm that the device is endorsed by the manufacturer supply chain. The policy may require that the relevant claims must have a match to a registered reference value. All claims may be worthy of additional appraisal. It is likely that most deployments would include a policy with appraisal for the following claims:

* Implementation ID - the value of the Implementation ID can be used to identify the verification requirements of the deployment.

* Software Component, Measurement Value - this value can uniquely identify a firmware release from the supply chain. In some cases, a Verifier may maintain a record for a series of firmware releases, being patches to an original baseline release. A verification policy may then allow this value to match any point on that release sequence or expect some minimum level of maturity related to the sequence.

* Software Component, Signer ID - where present in a deployment, this could allow a Verifier to operate a more general policy than that for Measurement Value as above, by allowing a token to contain any firmware entries signed by a known Signer ID, without checking for a uniquely registered version.

7.1. AR4SI Trustworthiness Claims Mappings

[RATS-AR4SI] defines an information model that Verifiers can employ to produce Attestation Results. AR4SI provides a set of standardized appraisal categories and tiers that greatly simplifies the task of writing Relying Party policies in multi-attester environments.

The contents of Table 3 are intended as guidance for implementing a PSA Verifier that computes its results using AR4SI. The table describes which PSA Evidence claims (if any) are related to which AR4SI trustworthiness claim, and therefore what the Verifier must consider when deciding if and how to appraise a certain feature associated with the PSA Attester.

Trustworthiness Vector claims	Related PSA claims
configuration	Software Components (Section 4.6.1)
executables	ditto
file-system	N/A
hardware	Implementation ID (Section 4.4.2) and CCA Platform config (TODO)
instance-identity	Instance ID (Section 4.4.1). The Security Lifecycle (Section 4.5.2) can also impact the derived identity.
runtime-opaque	Indirectly derived from executables, hardware, and instance-identity. The Security Lifecycle (Section 4.5.2) can also be relevant: for example, any debug state will expose otherwise protected memory.
sourced-data	N/A
storage-opaque	Indirectly derived from executables, hardware, and instance-identity.

Table 3: AR4SI Claims mappings

This document does not prescribe what value must be chosen based on each possible situation: when assigning specific Trustworthiness Claim values, an implementation is expected to follow the algorithm described in Section 2.3.3 of [RATS-AR4SI].

7.2. Endorsements, Reference Values and Verification Key Material

The [CCA-ENDORSEMENTS] defines a protocol based on the [RATS-CoRIM] data model that can be used to convey CCA Endorsements, Reference Values and Verification Key Material to the Verifier.

8. Implementation Status

// RFC Editor: please remove this section before publication.

Implementations of this specification are provided by the Trusted Firmware-RMM project [TF-RMM] and the Veraison project [Veraison]. These implementations are released as open-source software.

9. Security and Privacy Considerations

This specification re-uses the EAT specification and therefore the CWT specification. Hence, the security and privacy considerations of those specifications apply here as well.

Attestation tokens contain information that may be unique to a device and therefore they may allow singling out an individual device for tracking purposes. Deployments that have privacy requirements must take appropriate measures to ensure that the token is only used to provision anonymous/pseudonym keys.

10. IANA Considerations

10.1. CBOR Web Token Claims Registration

IANA is requested to make permanent the following claims that have been assigned via early allocation in the "CBOR Web Token (CWT) Claims" registry [IANA-CWT].

10.1.1. Security Lifecycle Claim

* Claim Name: arm-platform-security-lifecycle

* Claim Description: Arm Platform Security Lifecycle

* JWT Claim Name: N/A

* Claim Key: 2395

* Claim Value Type(s): unsigned integer

* Change Controller: Hannes Tschofenig TODO: Issue #34 (https://github.com/SimonFrost-Arm/draft-ffm-rats-cca-token/issues/34) find document centric change controller

* Specification Document(s): Section 4.5.2 of RFCthis

10.1.2. Implementation ID Claim

* Claim Name: arm-platform-implementation-id

* Claim Description: Arm Platform Implementation ID

* JWT Claim Name: N/A
* Claim Key: 2396
* Claim Value Type(s): byte string
* Change Controller: Hannes Tschofenig
* Specification Document(s): Section 4.4.2 of RFCthis

10.1.3. Software Components Claim

* Claim Name: arm-platform-software-components
* Claim Description: Arm Platform Software Components
* JWT Claim Name: N/A
* Claim Key: 2399
* Claim Value Type(s): array
* Change Controller: Hannes Tschofenig
* Specification Document(s): Section 4.6.1 of RFCthis

10.1.4. Verification Service Indicator Claim

* Claim Name: arm-platform-verification-service-indicator
* Claim Description: Arm Platform Verification Service Indicator
* JWT Claim Name: N/A
* Claim Key: 2400
* Claim Value Type(s): text string
* Change Controller: Hannes Tschofenig
* Specification Document(s): Section 4.7.1 of RFCthis

10.1.5. Platform Config Claim

* Claim Name: arm-platform-config
* Claim Description: Arm Platform Configuration

* JWT Claim Name: N/A
* Claim Key: 2401
* Claim Value Type(s): byte string
* Change Controller: Hannes Tschofenig
* Specification Document(s): Section 4.5.3 of RFCthis

10.1.6. Platform Hash Algorithm ID Clain

* Claim Name: arm-platform-hash-algm-id
* Claim Description: Arm Platform Hash Algorithm ID
* JWT Claim Name: N/A
* Claim Key: 2402
* Claim Value Type(s): text string
* Change Controller: Hannes Tschofenig
* Specification Document(s): Section 4.7.2 of RFCthis

10.1.7. CCA Token Platform Token Label

* Claim Name: cca-platform-token-label
* Claim Description: CCA Token Platform Token Label
* JWT Claim Name: N/A
* Claim Key: 44234
* Claim Value Type(s): byte string
* Change Controller: Hannes Tschofenig
* Specification Document(s): Section 4.1 of RFCthis

10.1.8. Realm Personalization Value Claim

* Claim Name: cca-realm-personalization-value
* Claim Description: CCA Realm Personalisation Value

* JWT Claim Name: N/A
* Claim Key: 44235
* Claim Value Type(s): byte string
* Change Controller: Hannes Tschofenig
* Specification Document(s): Section 4.8.3 of RFCthis

10.1.9. Realm Hash Algorithm ID Claim

* Claim Name: cca-realm-hash-algm-id
* Claim Description: CCA Realm Hash Algorithm ID
* JWT Claim Name: N/A
* Claim Key: 44236
* Claim Value Type(s): text string
* Change Controller: Hannes Tschofenig
* Specification Document(s): Section 4.8.6 of RFCthis

10.1.10. Realm Public Key Claim

* Claim Name: cca-realm-public-key
* Claim Description: CCA Realm Public Key
* JWT Claim Name: N/A
* Claim Key: 44237
* Claim Value Type(s): byte string
* Change Controller: Hannes Tschofenig
* Specification Document(s): Section 4.8.7 of RFCthis

10.1.11. Realm Initial Measurement Claim

* Claim Name: cca-realm-initial-measurement
* Claim Description: CCA Realm Initial Measurement

* JWT Claim Name: N/A
* Claim Key: 44238
* Claim Value Type(s): byte string
* Change Controller: Hannes Tschofenig
* Specification Document(s): Section 4.8.4 of RFCthis

10.1.12. Realm Extensible Measurements Claim

* Claim Name: cca-realm-extensible-measurements
* Claim Description: CCA Realm Extensible Measurements
* JWT Claim Name: N/A
* Claim Key: 44239
* Claim Value Type(s): array
* Change Controller: Hannes Tschofenig
* Specification Document(s): Section 4.8.4 of RFCthis

10.1.13. Realm Public Key Hash Algorithm ID Claim

* Claim Name: cca-realm-public-key-hash-algm-id
* Claim Description: Realm Public Key Hash Algorithm ID Claim
* JWT Claim Name: N/A
* Claim Key: 44240
* Claim Value Type(s): text string
* Change Controller: Hannes Tschofenig
* Specification Document(s): Section 4.8.8 of RFCthis

10.1.14. CCA Token Delegated Realm Token Label

* Claim Name: cca-platform-delegated-realm-label
* Claim Description: CCA Token Platform Token Label

* JWT Claim Name: N/A

* Claim Key: 44241

* Claim Value Type(s): byte string

* Change Controller: Hannes Tschofenig

* Specification Document(s): Section 4.1 of RFCthis

10.2. Media Types

No new media type registration is requested. To indicate that the transmitted content is a CCA attestation token, applications can use the application/eat+cwt media type defined in [EAT-MEDIATYPES] with the eat_profile parameter set to tag:arm.com,2023:cca_platform#1.0.0.

10.3. CoAP Content-Formats Registration

IANA is requested to register a CoAP Content-Format ID in the "CoAP Content-Formats" registry [IANA-CoAP-Content-Formats]:

* A registration for the application/eat+cwt media type with the eat_profile parameter equal to "tag:arm.com,2023:cca_platform#1.0.0"

The Content-Formats should be allocated from the Expert review range (0-255).

10.3.1. Registry Contents

* Media Type: ‘application/eat+cwt; eat_profile="tag:arm.com,2023:cca_platform#1.0.0"

* Encoding: -

* Id: To-be-assigned by IANA

* Reference: RFCthis

* Media Type: ‘application/eat+cwt; eat_profile="tag:arm.com,2023:realm#1.0.0"

* Encoding: -

* Id: To-be-assigned by IANA

* Reference: RFCthis

11. References

11.1. Normative References

[CCA-ARCH] Arm, "Learn the architecture - Introducing Arm Confidential Compute Architecture", May 2023, <https://developer.arm.com/documentation/den0125/0300>.

[CMW] Birkholz, H., Smith, N., Fossati, T., Tschofenig, H., and D. Glaze, "RATS Conceptual Messages Wrapper (CMW)", Work in Progress, Internet-Draft, draft-ietf-rats-msg-wrap-17, 12 August 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-rats-msg-wrap-17>.

[COSE-ALGS] Schaad, J., "CBOR Object Signing and Encryption (COSE): Initial Algorithms", RFC 9053, DOI 10.17487/RFC9053, August 2022, <https://www.rfc-editor.org/rfc/rfc9053>.

[EAT] Lundblade, L., Mandyam, G., O’Donoghue, J., and C. Wallace, "The Entity Attestation Token (EAT)", Work in Progress, Internet-Draft, draft-ietf-rats-eat-31, 6 September 2024, <https://datatracker.ietf.org/doc/html/draft-ietf-rats-eat-31>.

[EAT-MEDIATYPES] Lundblade, L., Birkholz, H., and T. Fossati, "EAT Media Types", Work in Progress, Internet-Draft, draft-ietf-rats-eat-media-type-12, 3 November 2024, <https://datatracker.ietf.org/doc/html/draft-ietf-rats-eat-media-type-12>.

[IANA-CWT] IANA, "CBOR Web Token (CWT) Claims", 2022, <https://www.iana.org/assignments/cwt/cwt.xhtml#claims-registry>.

[IANA.named-information] IANA, "Named Information", <https://www.iana.org/assignments/named-information>.

[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/rfc/rfc2119>.

[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/rfc/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/rfc/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/rfc/rfc8610>.

[RME] Arm, "Learn the architecture - Realm Management Extension", June 2021, <https://developer.arm.com/documentation/den0126/0100>.

[RMM] Arm, "Realm Management Monitor specification 1.0", December 2022, <https://developer.arm.com/documentation/den0137/1-0bet2>.

[STD94] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", STD 94, RFC 8949, DOI 10.17487/RFC8949, December 2020, <https://www.rfc-editor.org/rfc/rfc8949>.

[STD96] Schaad, J., "CBOR Object Signing and Encryption (COSE): Structures and Process", STD 96, RFC 9052, DOI 10.17487/RFC9052, August 2022, <https://www.rfc-editor.org/rfc/rfc9052>.

11.2. Informative References

[CCA-ENDORSEMENTS]
Deshpande, Y. and T. Fossati, "A CoRIM Profile for Arm’s Confidential Computing Architecture (CCA) Endorsements", Work in Progress, Internet-Draft, draft-ydb-rats-cca-endorsements-02, 4 July 2025, <https://datatracker.ietf.org/doc/html/draft-ydb-rats-cca-endorsements-02>.

[COSE-X509]
Schaad, J., "CBOR Object Signing and Encryption (COSE): Header Parameters for Carrying and Referencing X.509 Certificates", RFC 9360, DOI 10.17487/RFC9360, February 2023, <https://www.rfc-editor.org/rfc/rfc9360>.

[I-D.kdyxy-rats-tdx-eat-profile]
Kostal, G., Dittakavi, S., Yeluri, R., Xia, H., and J. Yu, "EAT profile for Intel(r) Trust Domain Extensions (TDX) attestation result", Work in Progress, Internet-Draft,

draft-kdyxy-rats-tdx-eat-profile-02, 13 December 2024, <https://datatracker.ietf.org/doc/html/draft-kdyxy-rats-tdx-eat-profile-02>.

[I-D.mandyam-rats-qwestoken]
Mandyam, G., Sekhar, V., and S. Mohammed, "The Qualcomm Wireless Edge Services (QWES) Attestation Token", Work in Progress, Internet-Draft, draft-mandyam-rats-qwestoken-00, 1 November 2019, <https://datatracker.ietf.org/doc/html/draft-mandyam-rats-qwestoken-00>.

[IANA-CoAP-Content-Formats]
IANA, "CoAP Content-Formats", 2022, <https://www.iana.org/assignments/core-parameters>.

[RATS-AR4SI]
Voit, E., Birkholz, H., Hardjono, T., Fossati, T., and V. Scarlata, "Attestation Results for Secure Interactions", Work in Progress, Internet-Draft, draft-ietf-rats-ar4si-09, 15 August 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-rats-ar4si-09>.

[RATS-CoRIM]
Birkholz, H., Fossati, T., Deshpande, Y., Smith, N., and W. Pan, "Concise Reference Integrity Manifest", Work in Progress, Internet-Draft, draft-ietf-rats-corim-08, 7 July 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-rats-corim-08>.

[RFC9334] Birkholz, H., Thaler, D., Richardson, M., Smith, N., and W. Pan, "Remote ATtestation procedureS (RATS) Architecture", RFC 9334, DOI 10.17487/RFC9334, January 2023, <https://www.rfc-editor.org/rfc/rfc9334>.

[TF-RMM] Trusted Firmware Project, "Trusted Firmware-RMM", 2022, <https://www.trustedfirmware.org/projects/tf-rmm>.

[Veraison] The Veraison Project, "Veraison ccatoken package", 2022, <https://github.com/veraison/ccatoken>.

Appendix A. Examples

The following examples show CCA attestation tokens for an hypothetical system comprising a single number of software component. The attesting device is in a lifecycle state (Section 4.5.2) of SECURED.

A.1. Delegated Mode

The following sample claim set and token are representative of a CCA Token using "delegated mode" described in Section 3.2.

In this model, the eat_nonce claim in the Platform token contains a hash of the RAK public key claim in the Realm token.

A.1.1. Platform Claims Set

The CCA Platform claims set is

```
=============== NOTE: '\' line wrapping per RFC 8792 ================

{
  265:"tag:arm.com,2023:cca_platform#1.0.0",
  10:h'\
   0D22E08A98469058486318283489BDB36F09DBEFEB1864DF433FA6E54EA2D711',
  2396:h'\
   7F454C4602010100000000000000000003003E00010000005058000000000000',
  256:h'\
 0107060504030201000F0E0D0C0B0A090817161514131211101F1E1D1C1B1A1918',
  2401:h'CFCFCFCF',
  2395:12291,
  2402:"sha-256",
  2400:"https://veraison.example/.well-known/veraison/verification",
  2399:[
    {
      1:"RSE_BL1_2",
      5:h'\
   5378796307535DF3EC8D8B15A2E2DC5641419C3D3060CFE32238C0FA973F7AA3',
      2:h'\
   9A271F2A916B0B6EE6CECB2426F0B3206EF074578BE55D9BC94F6F3FE3AB86AA',
      6:"sha-256"
    },
    {
      1:"RSE_BL2",
      5:h'\
   5378796307535DF3EC8D8B15A2E2DC5641419C3D3060CFE32238C0FA973F7AA3',
      2:h'\
   53C234E5E8472B6AC51C1AE1CAB3FE06FAD053BEB8EBFD8977B010655BFDD3C3',
      6:"sha-256"
    },
    {
      1:"RSE_S",
      5:h'\
   5378796307535DF3EC8D8B15A2E2DC5641419C3D3060CFE32238C0FA973F7AA3',
      2:h'\
```

```
      1121CFCCD5913F0A63FEC40A6FFD44EA64F9DC135C66634BA001D10BCF4302A2',
         6:"sha-256"
       },
       {
         1:"AP_BL1",
         5:h'\
      5378796307535DF3EC8D8B15A2E2DC5641419C3D3060CFE32238C0FA973F7AA3',
         2:h'\
      1571B5EC78BD68512BF7830BB6A2A44B2047C7DF57BCE79EB8A1C0E5BEA0A501',
         6:"sha-256"
       },
       {
         1:"AP_BL2",
         5:h'\
      5378796307535DF3EC8D8B15A2E2DC5641419C3D3060CFE32238C0FA973F7AA3',
         2:h'\
      10159BAF262B43A92D95DB59DAE1F72C645127301661E0A3CE4E38B295A97C58',
         6:"sha-256"
       },
       {
         1:"SCP_BL1",
         5:h'\
      5378796307535DF3EC8D8B15A2E2DC5641419C3D3060CFE32238C0FA973F7AA3',
         2:h'\
      10122E856B3FCD49F063636317476149CB730A1AA1CFAAD818552B72F56D6F68',
         6:"sha-256"
       },
       {
         1:"SCP_BL2",
         5:h'\
      F14B4987904BCB5814E4459A057ED4D20F58A633152288A761214DCD28780B56',
         2:h'\
      AA67A169B0BBA217AA0AA88A65346920C84C42447C36BA5F7EA65F422C1FE5D8',
         6:"sha-256"
       },
       {
         1:"AP_BL31",
         5:h'\
      5378796307535DF3EC8D8B15A2E2DC5641419C3D3060CFE32238C0FA973F7AA3',
         2:h'\
      2E6D31A5983A91251BFAE5AEFA1C0A19D8BA3CF601D0E8A706B4CFA9661A6B8A',
         6:"sha-256"
       },
       {
         1:"RMM",
         5:h'\
      5378796307535DF3EC8D8B15A2E2DC5641419C3D3060CFE32238C0FA973F7AA3',
         2:h'\
```

```
      A1FB50E6C86FAE1679EF3351296FD6713411A08CF8DD1790A4FD05FAE8688164',
         6:"sha-256"
       },
       {
         1:"HW_CONFIG",
         5:h'\
      5378796307535DF3EC8D8B15A2E2DC5641419C3D3060CFE32238C0FA973F7AA3',
         2:h'\
      1A252402972F6057FA53CC172B52B9FFCA698E18311FACD0F3B06ECAAEF79E17',
         6:"sha-256"
       },
       {
         1:"FW_CONFIG",
         5:h'\
      5378796307535DF3EC8D8B15A2E2DC5641419C3D3060CFE32238C0FA973F7AA3',
         2:h'\
      9A92ADBC0CEE38EF658C71CE1B1BF8C65668F166BFB213644C895CCB1AD07A25',
         6:"sha-256"
       },
       {
         1:"TB_FW_CONFIG",
         5:h'\
      5378796307535DF3EC8D8B15A2E2DC5641419C3D3060CFE32238C0FA973F7AA3',
         2:h'\
      238903180CC104EC2C5D8B3F20C5BC61B389EC0A967DF8CC208CDC7CD454174F',
         6:"sha-256"
       },
       {
         1:"SOC_FW_CONFIG",
         5:h'\
      5378796307535DF3EC8D8B15A2E2DC5641419C3D3060CFE32238C0FA973F7AA3',
         2:h'\
      E6C21E8D260FE71882DEBDB339D2402A2CA7648529BC2303F48649BCE0380017',
         6:"sha-256"
       }
     ]
   }
```


A.1.2. Realm Claims Set

The CCA Realm claims set is

```
=============== NOTE: '\' line wrapping per RFC 8792 ================

{
    265: "tag:arm.com,2023:realm#1.0.0", / eat_profile /
    10: h'\
6E86D6D97CC713BC6DD43DBCE491A6B40311C027A8BF85A39DA63E9CE44C132A8A11\
9D296FAE6A6999E9BF3E4471B0CE01245D889424C31E89793B3B1D6B1504', / \
                                                           eat_nonce /
    44236: "sha-256", / Realm hash algorithm /
    44240: "sha-256", / RAK hash algorithm /
    44235: h'\
546865207175696636B2062726F776E20666F78206A756D7073206F76657220313320\
6C617A7920646F67732E546865207175696636B2062726F776E20666F7820', / PV /
    44237: << { / RAK /
        1: 2, / kty=EC2 /
        -1: 2, / crv=P-384 /
        -2: h'\
76F988091BE585ED41801AECFAB858548C63057E16B0E676120BBD0D2F9C29E056C5\
                      D41A0130EB9C21517899DC23146B', / x-coordinate /
        -3: h'\
28E1B062BD3EA4B315FD219F1CBB528CB6E74CA49BE16773734F61A1CA61031B2BBF\
                       3D918F2F94FFC4228E50919544AE' / y-coordinate /
    } >>,
    44238: h'\
311314AB73620350CF758834AE5C65D9E8C2DC7FEBE6E7D9654BBE864E300D49', \
                                                               / RIM /
    44239: [
        h'\
24D5B0A296CC05CBD8068C5067C5BD473B770DDA6AE082FE3BA30ABE3F9A6AB1', \
                                                            / REM[0] /
        h'\
788FC090BFC6B8ED903152BA8414E73DAF5B8C7BB1E79AD502AB0699B659ED16', \
                                                            / REM[1] /
        h'\
DAC46A58415DC3A00D7A741852008E9CAE64F52D03B9F76D76F4B3644FEFC416', \
                                                            / REM[2] /
        h'\
32C6AFC627E55585C03155359F331A0E225F6840DB947DD96EFAB81BE2671939'  \
                                                            / REM[3] /
    ],
    44243: "private"  / MEC policy /
}
```

A.1.3. Platform Attestation Key

The COSE Key representation of the Platform Attestation Key (PAK) used for creating the COSE Sign1 signature over the CCA Platform token is

```
=============== NOTE: '\' line wrapping per RFC 8792 ================

{
  / kty /  1: 2,  / EC2 /
  / crv / -1: 2,  / P-384 /
  / x-coordinate / -2: h'\
212867C52E2B9508B0A420A90560F394D2DFAA21BDD7514FF1A901AFE7E1F78BB11D\
                                       4E66F8A8A38AFA76AF6A31C4DE8C',
  / y-coordinate / -3: h'\
84CE2DAFC9964258B53FAD718774F45620D111B176E8318E1187DB0235A318D37BA5\
                                       97FEE80E0E4C762A12BCB3EA6ED4',
  / private key /  -4: h'\
8AC090C995869F61AC1358F02B021A26AB6EB386203AC735D7CE9855538B91F74C44\
                                        B0D580243EFB799A293DCBAA0899'
}
```

A.1.4. Realm Attestation Key

The COSE Key representation of the Realm Attestation Key (RAK) used for creating the COSE Sign1 signature over the CCA Realm token is

```
=============== NOTE: '\' line wrapping per RFC 8792 ================

{
  / kty /  1: 2,  / EC2 /
  / crv / -1: 2,  / P-384 /
  / x-coordinate / -2: h'\
76F988091BE585ED41801AECFAB858548C63057E16B0E676120BBD0D2F9C29E056C5\
                                       D41A0130EB9C21517899DC23146B',
  / y-coordinate / -3: h'\
28E1B062BD3EA4B315FD219F1CBB528CB6E74CA49BE16773734F61A1CA61031B2BBF\
                                       3D918F2F94FFC4228E50919544AE',
  / private key /  -4: h'\
2011C7F03CEE4325176E524F033C0CE1E21A76E6C1A4F0B839AA1DF61E0E8A5C8A05\
                                        740F9B69EFA7EB1A4185BD117F68'
}
```

A.1.5. Signed and Bound Assembly

The resulting CMW collection is

```
=============== NOTE: '\' line wrapping per RFC 8792 ================

907({
  44234: [
    263,
    << 18([
      h'A1013822',
```

```
         {},
         << {
           265:"tag:arm.com,2023:cca_platform#1.0.0",
           10:h'\
      0D22E08A98469058486318283489BDB36F09DBEFEB1864DF433FA6E54EA2D711',
           2396:h'\
      7F454C4602010100000000000000000003003E00010000005058000000000000',
           256:h'\
    0107060504030201000F0E0D0C0B0A090817161514131211101F1E1D1C1B1A1918',
           2401:h'CFCFCFCF',
           2395:12291,
           2402:"sha-256",
           2400:"https://veraison.example/.well-known/veraison/\
                                                          verification",
           2399:[
             {
               1:"RSE_BL1_2",
               5:h'\
      5378796307535DF3EC8D8B15A2E2DC5641419C3D3060CFE32238C0FA973F7AA3',
               2:h'\
      9A271F2A916B0B6EE6CECB2426F0B3206EF074578BE55D9BC94F6F3FE3AB86AA',
               6:"sha-256"
             },
             {
               1:"RSE_BL2",
               5:h'\
      5378796307535DF3EC8D8B15A2E2DC5641419C3D3060CFE32238C0FA973F7AA3',
               2:h'\
      53C234E5E8472B6AC51C1AE1CAB3FE06FAD053BEB8EBFD8977B010655BFDD3C3',
               6:"sha-256"
             },
             {
               1:"RSE_S",
               5:h'\
      5378796307535DF3EC8D8B15A2E2DC5641419C3D3060CFE32238C0FA973F7AA3',
               2:h'\
      1121CFCCD5913F0A63FEC40A6FFD44EA64F9DC135C66634BA001D10BCF4302A2',
               6:"sha-256"
             },
             {
               1:"AP_BL1",
               5:h'\
      5378796307535DF3EC8D8B15A2E2DC5641419C3D3060CFE32238C0FA973F7AA3',
               2:h'\
      1571B5EC78BD68512BF7830BB6A2A44B2047C7DF57BCE79EB8A1C0E5BEA0A501',
               6:"sha-256"
             },
             {
```

```
          1:"AP_BL2",
          5:h'\
5378796307535DF3EC8D8B15A2E2DC5641419C3D3060CFE32238C0FA973F7AA3',
          2:h'\
10159BAF262B43A92D95DB59DAE1F72C645127301661E0A3CE4E38B295A97C58',
          6:"sha-256"
        },
        {
          1:"SCP_BL1",
          5:h'\
5378796307535DF3EC8D8B15A2E2DC5641419C3D3060CFE32238C0FA973F7AA3',
          2:h'\
10122E856B3FCD49F063636317476149CB730A1AA1CFAAD818552B72F56D6F68',
          6:"sha-256"
        },
        {
          1:"SCP_BL2",
          5:h'\
F14B4987904BCB5814E4459A057ED4D20F58A633152288A761214DCD28780B56',
          2:h'\
AA67A169B0BBA217AA0AA88A65346920C84C42447C36BA5F7EA65F422C1FE5D8',
          6:"sha-256"
        },
        {
          1:"AP_BL31",
          5:h'\
5378796307535DF3EC8D8B15A2E2DC5641419C3D3060CFE32238C0FA973F7AA3',
          2:h'\
2E6D31A5983A91251BFAE5AEFA1C0A19D8BA3CF601D0E8A706B4CFA9661A6B8A',
          6:"sha-256"
        },
        {
          1:"RMM",
          5:h'\
5378796307535DF3EC8D8B15A2E2DC5641419C3D3060CFE32238C0FA973F7AA3',
          2:h'\
A1FB50E6C86FAE1679EF3351296FD6713411A08CF8DD1790A4FD05FAE8688164',
          6:"sha-256"
        },
        {
          1:"HW_CONFIG",
          5:h'\
5378796307535DF3EC8D8B15A2E2DC5641419C3D3060CFE32238C0FA973F7AA3',
          2:h'\
1A252402972F6057FA53CC172B52B9FFCA698E18311FACD0F3B06ECAAEF79E17',
          6:"sha-256"
        },
        {
```

```
               1:"FW_CONFIG",
               5:h'\
      5378796307535DF3EC8D8B15A2E2DC5641419C3D3060CFE32238C0FA973F7AA3',
               2:h'\
      9A92ADBC0CEE38EF658C71CE1B1BF8C65668F166BFB213644C895CCB1AD07A25',
               6:"sha-256"
             },
             {
               1:"TB_FW_CONFIG",
               5:h'\
      5378796307535DF3EC8D8B15A2E2DC5641419C3D3060CFE32238C0FA973F7AA3',
               2:h'\
      238903180CC104EC2C5D8B3F20C5BC61B389EC0A967DF8CC208CDC7CD454174F',
               6:"sha-256"
             },
             {
               1:"SOC_FW_CONFIG",
               5:h'\
      5378796307535DF3EC8D8B15A2E2DC5641419C3D3060CFE32238C0FA973F7AA3',
               2:h'\
      E6C21E8D260FE71882DEBDB339D2402A2CA7648529BC2303F48649BCE0380017',
               6:"sha-256"
             }
           ]
         } >>,
         h'\
   31D04D52CCDE952C1E32CBA181885A40B8CC38E0528C1E89589807642AA5E3F2BC37\
   F95374506BFF4D2E4BE7063C4D72419270C722E8D4D93EE8B6C9FACE3B43C9761A49\
               941AB6F38FFDFF496AD463B4CBFA11D83E23E31F7F62329DE30C1CC8'
       ]) >>
     ],

     44241: [
       263,
       << 18([
         h'A1013822',
         {},
         << {
           265:"tag:arm.com,2023:realm#1.0.0",
           10:h'\
   6E86D6D97CC713BC6DD43DBCE491A6B40311C027A8BF85A39DA63E9CE44C132A8A11\
          9D296FAE6A6999E9BF3E4471B0CE01245D889424C31E89793B3B1D6B1504',
           44236:"sha-256",
           44240:"sha-256",
           44235:h'\
   546865207175696636B2062726F776E20666F78206A756D7073206F76657220313320\
          6C617A7920646F67732E546865207175696636B2062726F776E20666F7820',
           44237:h'\
```

```
A40102200221583076F988091BE585ED41801AECFAB858548C63057E16B0E676120B\
BD0D2F9C29E056C5D41A0130EB9C21517899DC23146B22583028E1B062BD3EA4B315\
FD219F1CBB528CB6E74CA49BE16773734F61A1CA61031B2BBF3D918F2F94FFC4228E\
                                                         50919544AE’,
        44238:h’\
   311314AB73620350CF758834AE5C65D9E8C2DC7FEBE6E7D9654BBE864E300D49’,
        44239:[
          h’\
   24D5B0A296CC05CBD8068C5067C5BD473B770DDA6AE082FE3BA30ABE3F9A6AB1’,
          h’\
   788FC090BFC6B8ED903152BA8414E73DAF5B8C7BB1E79AD502AB0699B659ED16’,
          h’\
   DAC46A58415DC3A00D7A741852008E9CAE64F52D03B9F76D76F4B3644FEFC416’,
          h’\
    32C6AFC627E55585C03155359F331A0E225F6840DB947DD96EFAB81BE2671939’
        ],
        44243: "private"  / MEC policy /
      } >>,
      h’\
580B1DEA32D30AC6884C86B39CBE0FCB03BD00DF5103F9BAB01386A46A3BA8143E27\
ED6D4EB0D0A2724ABDF9640C09462FACE6DF186909DFA6EB131E3A7918276077ACDA\
            B8A8BDECA6B0EAAFAB66E1439C1371F4FB1D6AAC047481B5DC75DD46’
    ]) >>
  ]
})
```

which has the following base16 encoding:


```
d9038ba219acca821901075905eed28444a1013822a0590581a919010978
237461673a61726d2e636f6d2c323032333a6363615f706c6174666f726d
23312e302e300a58200d22e08a98469058486318283489bdb36f09dbefeb
1864df433fa6e54ea2d71119095c58207f454c4602010100000000000000
000003003e00010000005058000000000000190100582101070605040302
01000f0e0d0c0b0a090817161514131211101f1e1d1c1b1a191819096144
cfcfcfcf19095b19300319096267736861 2d323536190960783a68747470
733a2f2f7665726169736f6e2e6578616d706c652f2e77656c6c2d6b6e6f
776e2f7665726169736f6e2f766572696669636174696f6e19095f8da401
695253455f424c315f32055820537879630 7535df3ec8d8b15a2e2dc5641
419c3d3060cfe32238c0fa973f7aa30258209a271f2a916b0b6ee6cecb24
26f0b3206ef074578be55d9bc94f6f3fe3ab86aa06677368612d323536a4
01675253455f424c3205582053787963075 35df3ec8d8b15a2e2dc564141
9c3d3060cfe32238c0fa973f7aa302582053c234e5e8472b6ac51c1ae1ca
b3fe06fad053beb8ebfd8977b010655bfdd3c306677368612d323536a401
655253455f5305582053787963075 35df3ec8d8b15a2e2dc5641419c3d30
60cfe32238c0fa973f7aa30258201121cfccd5913f0a63fec40a6ffd44ea
64f9dc135c66634ba001d10bcf4302a206677368612d323536a401664150
5f424c31055820537879630 7535df3ec8d8b15a2e2dc5641419c3d3060cf
e32238c0fa973f7aa30258201571b5ec78bd68512bf7830bb6a2a44b2047
```

```
c7df57bce79eb8a1c0e5bea0a50106677368612d323536a4016641505f42
4c320558205378796307535df3ec8d8b15a2e2dc5641419c3d3060cfe322
38c0fa973f7aa302582010159baf262b43a92d95db59dae1f72c64512730
1661e0a3ce4e38b295a97c5806677368612d323536a401675343505f424c
3105582053787963075355df3ec8d8b15a2e2dc5641419c3d3060cfe32238
c0fa973f7aa30258201012 2e856b3fcd49f063636317476149cb730a1aa1
cfaad818552b72f56d6f6806677368612d323536a401675343505f424c32
055820f14b4987904bcb5814e4459a057ed4d20f58a633152288a761214d
cd28780b56025820aa67a169b0bba217aa0aa88a65346920c84c42447c36
ba5f7ea65f422c1fe5d806677368612d323536a4016741505f424c333105
58205378796307535df3ec8d8b15a2e2dc5641419c3d3060cfe32238c0fa
973f7aa30258202e6d31a5983a91251bfae5aefa1c0a19d8ba3cf601d0e8
a706b4cfa9661a6b8a06677368612d323536a40163524d4d055820537879
6307535df3ec8d8b15a2e2dc5641419c3d3060cfe32238c0fa973f7aa302
5820a1fb50e6c86fae1679ef3351296fd6713411a08cf8dd1790a4fd05fa
e868816406677368612d323536a4016948575f434f4e4649470558205378
796307535df3ec8d8b15a2e2dc5641419c3d3060cfe32238c0fa973f7aa3
0258201a252402972f6057fa53cc172b52b9ffca698e18311facd0f3b06e
caaef79e1706677368612d323536a4016946575f434f4e46494705582053
78796307535df3ec8d8b15a2e2dc5641419c3d3060cfe32238c0fa973f7a
a30258209a92adbc0cee38ef658c71ce1b1bf8c65668f166bfb213644c89
5ccb1ad07a2506677368612d323536a4016c54425f46575f434f4e464947
0558205378796307535df3ec8d8b15a2e2dc5641419c3d3060cfe32238c0
fa973f7aa3025820238903180cc104ec2c5d8b3f20c5bc61b389ec0a967d
f8cc208cdc7cd454174f06677368612d323536a4016d534f435f46575f43
4f4e4649470558205378796307535df3ec8d8b15a2e2dc5641419c3d3060
cfe32238c0fa973f7aa3025820e6c21e8d260fe71882debdb339d2402a2c
a7648529bc2303f48649bce0380017 06677368612d32353658 6031d04d52
ccde952c1e32cba181885a40b8cc38e0528c1e89589807642aa5e3f2bc37
f95374506bff4d2e4be7063c4d72419270c722e8d4d93ee8b6c9face3b43
c9761a49941ab6f38ffdff496ad463b4cbfa11d83e23e31f7f62329de30c
1cc819acd182190107590259d28444a1013822a05901eca9190109781c74
61673a61726d2e636f6d2c323032333a7265616c6d23312e302e300a5840
6e86d6d97cc713bc6dd43dbce491a6b40311c027a8bf85a39da63e9ce44c
132a8a119d296fae6a6999e9bf3e4471b0ce01245d889424c31e89793b3b
1d6b150419accc677368612d32353619acd0677368612d32353619accb58
405468652071756963 6b2062726f776e20666f78206a756d7073206f7665
72203133206c617a7920646f67732e54686520717569636b2062726f776e
20666f782019accd586ba4010220022158 3076f988091be585ed41801aec
fab858548c63057e16b0e676120bbd0d2f9c29e056c5d41a0130eb9c2151
7899dc23146b22583028e1b062bd3ea4b315fd219f1cbb528cb6e74ca49b
e16773734f61a1ca61031b2bbf3d918f2f94ffc4228e50919544ae19acce
5820311314ab73620350cf758834ae5c65d9e8c2dc7febe6e7d9654bbe86
4e300d4919accf84582024d5b0a296cc05cbd8068c5067c5bd473b770dda
6ae082fe3ba30abe3f9a6ab15820788fc090bfc6b8ed903152ba8414e73d
af5b8c7bb1e79ad502ab0699b659ed165820dac46a58415dc3a00d7a7418
52008e9cae64f52d03b9f76d76f4b3644fefc416582032c6afc627e55585
c03155359f331a0e225f6840db947dd96efab81be267193919acd3677072
```

69766174655860580b1dea32d30ac6884c86b39cbe0fcb03bd00df5103f9
bab01386a46a3ba8143e27ed6d4eb0d0a2724abdf9640c09462face6df18
6909dfa6eb131e3a7918276077acdab8a8bdeca6b0eaafab66e1439c1371
f4fb1d6aac047481b5dc75dd46

A.2. Direct Mode

The following sample claim sets and the resulting CCA Token are representative of a CCA Token using "direct mode" (Section 3.1).

In "direct mode" the eat_nonce claim in the Platform token contains a hash of the Realm claims set, which includes verifier-provided challenge data.

TODO

Acknowledgments

TODO

Contributors

Yogesh Deshpande
Arm Limited
Email: Yogesh.Deshpande@arm.com

Sergei Trofimov
Arm Limited
Email: Sergei.Trofimov@arm.com

Authors’ Addresses

Simon Frost
Arm Limited
Email: Simon.Frost@arm.com

Thomas Fossati
Linaro
Email: thomas.fossati@linaro.org

Giri Mandyam
Mediatek Inc
Email: giridhar.mandyam@gmail.com